AFIT/GEO/ENG/93D-04

Handwritten Word Recognition Based on Fourier Coefficients

THESIS

Gary Shartle
Captain, USAF

AFIT/GEO/ENG/93D-04

Approved for public release; distribution unlimited

AFIT/GEO/ENG/93D-04

Handwritten Word Recognition Based on Fourier Coefficients

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Gary Shartle, B.S.E.

Captain, USAF

December, 1993

Acknowledgements

The results of this research are the culmination of many factors. First and foremost, my lovely wife Deborah and my beautiful baby Erica Katherine allowed me to spend the time required to complete the research. Their loving inspiration sustained my pursuit. Secondly, Dr Rogers provided invaluable guidance and motivation. I cannot say enough thanks. Thirdly, all the committee members, Capt Ruck, Dr. Kabrisky, and Major Mark O'Hair, greatly assisted me thoughout the project, to whom I owe many thanks. And finally, without the assistance of the many friends here at AFIT, Bob MacDonald, John Keller, Neale Prescott, Kim McCrae, Martin Chin, Curtis Martin, Tom Burns, Ken Fielding, Dan Zambon, Dave Doak, Delores Bailey, and many others, this research would not be possible.

Gary Shartle

Table of Contents

List of Figures

List of Tables

Abstract

A machine which can read unconstrained words remains an unsolved problem. For example, automatic entry of handwritten documents into a computer is yet to be accomplished. Most systems attempt to segment letters of a word and read words one character at a time. Segmenting a handwritten word is very difficult and often, the confidence of the results is low. Another method which avoids segmentation altogether is to treat each word as a whole. This research investigates the use of Fourier Transform coefficients, computed from the whole word, for the recognition of handwritten words. To test this concept, the particular pattern recognition problem studied consisted of classifying four handwritten words, 'Buffalo', 'Vegas', 'Washington', 'City.' Several feature subsets of the Fourier coefficients are examined. The best recognition performance of 76.2% was achieved when the Karhunen-Loeve transform was computed on the Fourier coefficients.

Handwritten Word Recognition Based on Fourier Coefficients

I. Introduction

1.1 Background

Virtually any company or organization spending large sums of money processing documents is interested in developing a system to recognize unconstrained words, i.e. a reading machine. For example, the United States Postal Service funds research to develop an automated system for reading handwritten addresses on mail. Similarly, banking institutions want a system to read the handwritten amount on a check. Further, the census bureau, which processes approximately 250 million forms every ten years, is searching for an automated system to read the occupation block on their form. A system capable of reading handwritten words has tremendous value to the aforementioned organizations. Such a system would also fit in nicely with the needs of the Air Force because thousands of jobs are dedicated to document processing and the current Department of Defense policy is down-sizing the workforce.

Handwritten character and word recognition has been intensely investigated for the past 50 years. L.A. Pintsov states that although many algorithms have been developed to recognize handwritten characters, few new ideas on how to solve the problem have come about in the past thirty years (20). This comment indicates the difficulty in machine recognition of unconstrained text.

What is the best recognition that we can expect to achieve by a machine? The human recognition rate error rate for unconstrained isolated characters is about 4 percent, based on widely accepted experimental data (20). For unconstrained isolated words it may even be greater.

1.2 Problem Statement

In broad terms, a machine that can read unconstrained 'words' is undeveloped. Specifically, a machine's ability to read text in handwritten form by recognizing the entire

word is not yet possible. This research effort investigates the use of Fourier transform coefficients as unique features which can be used to classify a group of handwritten words.

1.3 Summary of Current Knowledge

Recognition of 5000 machine-typed words is possible, with 99% accuracy using Fourier coefficients, as demonstrated by O'Hair (16). The Service de Recherche Technique de la Poste has developed a device to recognize the handwritten amount on postal checks (9). Recognition rates reported 79% on a test set of 2492 words from 27 classes using Hidden Markov Models (9). No rejection rates were reported.

1.4 Research Objectives

This thesis will investigate a means to classify a group of handwritten words. The specific objectives of this research effort are as follows:

1. Develop a method of calculating features based on the 2-dimensional discrete Fourier Transform.

2. Investigate the use of the F-ratio for feature subset selection.

3. Investigate using Karhunen-Loeve transform as a means of feature set reduction.

4. Test several feature subsets of Fourier coefficients using the multi-layer perceptron and k-nearest neighbor.

5. Develop a method to incorporate an "add-on" feature selection procedure.

1.5 Approach and Methodology

This thesis will use data of handwritten words provided by the Center of Excellence for Document Analysis and Recognition, State University of New York at Buffalo (15). The database contains handwritten cities, states, ZIP codes, and others; however, only the handwritten cities will be used. The data is limited so only a four-class problem will be attempted. Additional data will be collected from people working in the laboratory. The training set and test set will include 200 patterns each.

A variety of feature extraction algorithms are examined to produce a suitable feature set for classification.

1.5.1 Fourier Analysis. The use of Fourier coefficients to classify whole, machine-typed, words has proven very successful in past research (17). This study will pursue the use of Fourier coefficients in the recognition of handwritten words. The thrust of the research will be searching for the best subset of features out of the Fourier coefficients calculated from the 2-dimensional discrete Fourier Transform.

1.5.2 Handwritten Word Classification. The primary classification techniques that will be used are the k-nearest neighbor and the multi-layer perceptron. The k-nearest neighbor classifier is a quick and easy way to classify a pattern. A sample is assigned to a class based on the class given by the k nearest neighbors (5). The multi-layer perceptron is a means to separate non-linearly separable data (23).

1.6 Thesis Organization

This thesis document is divided in the following way: Chapter II discusses past research in the area of whole word recognition. Chapter III describes the method of this study. Chapter IV reports the results of the study. Chapter V discusses the results and conclusions.

1.7 Summary

The recognition of machine-typed words has proven very successful (16); however, the recognition of handwritten words has had little success. This thesis will attack the difficult problem of machine recognition of unconstrained handwritten words based on features calculated from the Fourier Transform of the image of the word. Several methods of feature subset selection will be employed.

II. Literature Review

2.1 Introduction

This literature review examines the background information relevant to this study of handwritten word recognition. The current techniques used in handwritten word recognition are discussed as well as the justification for using the whole word to avoid character segmentation. Finally, a brief review of the classification techniques used in this study is presented.

2.2 Background

Much is known on where information is processed in the brain. The human visual system has been extensively mapped out (25). The visual system compresses visual images at a ratio of 100 to 1, which leads researchers to believe that only essential information is needed to construct a visual model of the world (23). The big problem is that it is not known exactly how the information is compressed. It has been theorized by Kabrisky that this compression can be modeled in part by computing the Fourier Transform of visual images (11). There is evidence that this may be exactly what is going on. (23, 2). A few examples show the correlation between Fourier distances and human psychological tests. The famous animal cracker test discussed in Dr. Rogers' book showed that the relative closeness of pairs of animals rated by humans corresponded to their respective distance in Fourier space (23). Similarly, Bush generated a better set of characters for pilots to see, especially under stressful conditions (2). By using coefficients computed from the Fourier Transform, he compared the nearest neighbor distance in Fourier space to the letters that were misclassified by humans and discovered the mistakes related to the distance. When changing the font style to spread out the distance in Fourier space, less mistakes were made by humans (2). Further, Ginsburg showed that many visual illusions work not only for humans, but also for the Fourier Transform (23).

Although every attempt is made to simulate what is happening in the human brain, one must face the fact that at this point, all one can do is attempt to classify patterns based on numbers. As Rogers says in his book, "Since nobody understands how brains

do any processing of significance, statements that these artificial neural networks work as brains do are Lies, Lies, all Lies!!" (23).

2.3 Difficulty in Segmenting Characters of Words

Trying to separate the individual characters of a word is extremely difficult when the word is handwritten, especially handwritten script. Any person can easily read this text and pick out the characters that make up each word. Even neatly written cursive script can easily be read. But those tasks are very difficult for a computer or machine to do. For example, in typewritten text of the type found in magazines, when an 'r' and 'n' are side by side, the machine may confuse the word 'modern' for 'modem' (24). Another difficulty could occur when non-text information appears in the image (3). For instance, if a word or group of words is underlined, such as the title of a book, a machine could mistake the underlining as part of each character. If the word 'The' is in the title and underlined, the machine could mistake the 'T' for and 'I'. Finally, a machine could encounter difficulties in reading text when letters overlap in terms of their defined space, an occurrence known as kerning. Letter overlap commonly occurs in cursive or italic writing where a capital letter is followed by a lower case letter that sits directly beneath a portion of the capitalized letter. Therefore, the question was asked, is there any way to avoid these difficulties?

2.4 Current Methods of Handwritten Word Recognition

In his 1985 thesis, O'Hair proposed treating words as single symbols to avoid the difficult segmentation problem of separating each word into its individual characters (16). His experiments were tested on typewritten text. By collecting features of the words from the lower harmonics of the 2-dimensional discrete Fourier transform and classifying them with the k-nearest neighbor algorithm, he achieved a recognition rate of 94%. He continued this work, expanding his database of typewritten words to include words using all fonts and various character separations. Several distance metrics were used to measure the features against a template of the word. He achieved a 99% recognition rate on 5,000 words (17).

Srihari uses two methods for word recognition (27). One of the methods is based on segmenting the word into characters and identifying the characters (27). Using this

method, they achieved a 92.0% recognition rate on a ten class problem with 3,000 words (27). The second method they used is to break the word up into segments and recognize each segment using Hidden Markov Models (27). The recognition rate on the same set of data in 93% (27). They also performed tests interpreting handwritten address on mail at a current performance of 44% with 6% error (27).

Tin Kam Ho, et al. experimented with machine-typed whole word recognition (10). They used a combination of features based on character segmentation and the use of the whole word (10). This was one of the first attempts at using a combination of techniques to recognize words (10). The features they used in the the whole word analysis were based on using a 7x7 template and convolving with the image. This resulted in a 1280-dimensional feature vector. Secondly, stroke direction was used, resulting in a 160-dimensional feature vector. Four types of strokes directions were used; east-west, northeast-southwest, north-south, and northwest-southeast. Then a nearest neighbor classifier was used to classify the word. This decision was then combined with results from five other independent classifiers to get a consensus ranking. The experimental results showed a 88.9% recognition rate with 1671 words in a 33850 word lexicon. This is a good example of the use of combining independent classifiers.

Researchers in France are currently developing a real world system to read the amount on a postal check (13). They compare the handwritten word amount to the handwritten digit amount to verify the amount on the check. In the recognition of the handwritten word amount, they fuse two methods for recognition. The two methods are based on whole word recognition and character segmentation of the whole word into characters. The whole word method uses dynamic time-warping. By looking for vertical lines, loops, horizontal lines, and dots, they built a representation of the word by which to compare to the reference codebook words. The fusion process compares the output classification of the whole word method to the character segmentation method. In the final process, the digit amount is compared to the handwritten amount and the final amount is determined. As of 1991, the recognition rate was 40% on a set of 6,400 samples. This example illustrates the difficulty of handwritten word recognition on real world data.

Some research has been done in trying to characterize the variability in cursive handwriting (21). Some of the characteristics examined were dots, dashes, writing slant, zones and baselines, zone heights, local min and max, concavity and curvature, loops or spikes, cusps, and the way letters are connected. The conclusions were that it is very difficult to characterize styles of handwriting. Recognition rates for four different styles of handwriting resulted in 89.5%, 82.0%, 49.0%, and 21.0%. Again, trying to find the appropriate features continues to elude researchers today.

Hidden Markov Models, which have been successful in speech recognition are now being applied to handwritten word recognition. Hidden Markov Models are effective in lowering the sensitivity to many variations in styles of handwriting (7). This is another technique that can avoid the difficult segmentation stage (7). Bertille and Yacoubi used Hidden Markov Models to recognize postal codes without segmentation (1). They achieved a range of recognition rates from 28% to 59% for 14 and 16 states respectively (1). Other research which uses Hidden Markov Models is reported. Gilloux, who heads the Service de Recherche Technique de la Poste, developed a prototype system to recognize the handwritten amount on a check (9). The words are segmented into segments which are used for recognition using Hidden Markov Models. The results reported a 79% recognition rate on a test set of 2492 words of 27 classes (9). They also reported that the system had poor generalization capabilities. (9)

All the methods described so far involve using some type of algorithm or combination of algorithms. J.C. Simon describes several principles which allow for more robust algorithms in recognizing handwritten words (26). The principles he applies are as follows: 1) several levels of recognition where each level is assigned a probability of occurrence, 2) use independent sources of information and estimate joint performance, and 3) feedback is utilized between each level of recognition (26). These principles are based on neurophysiological evidence (26). The results reported applying these principles showed a 79.5% recognition rate on a lexicon of 25 words (26).

2.5 Feature Extraction

All pattern recognition people will say that good features makes for good recognition. Finding features that distinguish one pattern from another is the key to a viable pattern recognition system. "Most papers give no reasons for the choice of features. In fact, most features in pattern recognition work are chosen on the grounds that the choice is intuitively reasonable. Once the features have been chosen authors apply sophisticated statistical methods in order to minimize errors. In most cases, however, the game has been lost with the choice of features." (20). "There is an unfortunate tendency in many articles describing laboratory systems to describe the classification method in detail, and to give inadequate or no information about the features used as the basis for classification, how they are measured, and most importantly, how they were derived and why they were chosen over other feature metrics (20)." Features are not usually defined in terms of style variations or distortion (20). It seems reasonable that much effort should be placed in developing good features. Of course, this is the hard part and why this problem has been around for so long. Especially when you have people believing that, "There is no unique computational procedure which can "extract" the identity of a character from its image."(20) If this is true, then computing the Fourier transform will be of no value. That is what this study intends to find out.

If it is true that no computational procedure can extract the identity of a character or a word, then the other option is to use the raw data. According to Fukunaga, "... as long as features are computed from the measurements, the set of features cannot carry more classification information than the measurements." (6) Raw measurements in the case of the image of a word are raw pixel values. These pixel values become the features.

Pintsov discusses two types of models which can describe a character. The first, called generative, is thinking of the path that a writing instrument follows when generating a character. How the character is formed is what can be analyzed. The second, called transformative, believes in an ideal form of the character, so the shape itself is analyzed in this case. It is not known exactly what the ideal character form is. Because there is no formal definition of a character, one must rely on data of human recognition and

perception. Points where the contour changes direction is much more informative than points on flat portions.

What happens in our visual system when we see an image? When looking at words, or anything for that matter, the brain's neurons are firing to produce the illusion in the brain of the scene being viewed. The image falling on the retina is compressed by a factor of 100:1 as it leaves the retina and this data is enough information to construct the word in the brain (23). At this point, the compressed data is the feature set used by the brain. Now then, the image that falls on the retina may contain other information that is not part of the word being looked at. So the actual compression of the word may not be 100:1, but it certainly is reduced to some amount. For a computer, at this point, the image of the word is represented by an array of pixel values and no other information. If we are to compress this information, how do we do it? A starting place may be to binarize the image.

Recognizing a handwritten word is similar to recognizing say a hand or foot or nose, even though there are all different types of variations, the object or word is the same. The brain somehow extracts constant features. The primary visual cortex processes color, form, and motion in separate areas (25) So there is more information contained in a handwritten word, such as color and whether it is moving or not.

If we are searching for features based on evidence of features used by humans and animals, then, "The obvious place to start is to emulate the ruthless preprocessing that is accomplished by our biological sensors." (23)

2.6 Previous Success using Fourier Features

A lot of work has been done in the field of pattern recognition using Fourier coefficients as features, especially trying to classify machine printed numbers, letters, and words. Radoy achieved success trying to recognize machine generated letters of the alphabet (22). The features he used to classify the letters came from computing the 2 dimensional discrete Fourier Transform. Tallman tried to relate the classification of visual images by the human visual system to a digital simulation based on spatial filtering (29). He filtered

out the low order Fourier frequency components and was able to recognize handwritten letters of the alphabet with a 95.8 percent accuracy using the 7 × 7 low pass frequency filter. And more recently, O'Hair achieved a successful recognition rate by classifying 5000 machine-printed words using low order Fourier coefficients (17). The continued success of the Fourier Transform in classifying letters of the alphabet and machine-typed words makes the Fourier Transform an obvious choice to use on handwritten words.

2.7 Classification Techniques

2.7.1 k-nearest neighbor. The k-nearest neighbor method for classification uses the rule that a sample is assigned a class based on the class given by the k nearest neighbors (5) In this study, each pattern will have a spot in Fourier space. A Euclidean distance is calculated to find the closest neighbor.

The program used to run a k-nn neighbor algorithm is called LNKnet (12). The training portion of the program stores all the input patterns. The testing portion of the program computes the Euclidean distance from the input pattern to all the stored patterns. "The class selected for the test pattern is the one with a plurality of the classes of the k nearest neighbors. (12)" Appendix A contains a tutorial for using LNKnet.

2.7.2 multi-layer perceptron. The multi-layer perceptron is a method that develops a non-linear discriminant function during training that separates training data. The architecture of the network consists of three layers; an input layer, hidden layer, and output layer. The input layer has as many inputs as the number of features for each pattern, while the number of output nodes is determined by the number of classes to be recognized. The weighted sums of the input layer is passed through a sigmoid function at each hidden node, likewise the weighted sum of the hidden layer is passed through a sigmoid function at each output node. During training, the weights are updated to provide the minimum error at the output layer. A gradient descent method is used to update the weights through back propagation. For testing, the class corresponding to the highest output node value is determined the class of the pattern (12).

2.7.3 Fusion of Classifiers. The state of the art pattern recognition techniques used in Japan today are based on what they call "multi-expert" recognition (pg 8). Multi-expert recognition enhances the overall recognition by combining several independent methods of recognition. A character recognition competition was held in 1992 and the highest three scoring algorithms were combined and used to classify the test set used during the competition. The test set included 10,000 samples of the digits 0-9. The results showed an improvement from 96.2% to 99%.

Guerts reported a 4.3% increase in recognition of targets by fusing independent classifiers acting on the same data.

No matter how fancy the classification technique though, the features are the most important part. Without the right features, the best classification routine won't work.

2.8 Conclusion

This chapter of the thesis reviewed current techniques in handwritten word recognition and the past success of Fourier coefficients in recognizing machine-typed text. It is the goal of this study to determine whether or not it is feasible to use Fourier coefficients computed from handwritten words for classification.

III. Approach and Methodology

3.1 Introduction

This part of the thesis describes the method used in this study to determine whether or not the Fourier coefficients computed from the word images are of value in classifying hand-printed words. A description of the data used, how it was preprocessed, the method of feature extraction, and the type of classification used encompasses this chapter.

3.2 Data Set Description

3.2.1 Handwritten Words. In general, it is very difficult to obtain real world data for pattern recognition. It is especially difficult trying to find a good set of handwritten words for an experiment. The State University of New York at Buffalo produced a database of handwritten cities, states, and Zip codes. It is from this database that some of the handwritten words used in this study were obtained.

To further describe the difficulties in obtaining data, the database contains over 3000 handwritten words, yet, few examples of any particular word existed in the database; therefore, the words with the most occurrences were chosen. The words 'buffalo', 'city', 'washington', and 'vegas' topped the list of the most examples. These examples consisted of printed as well as script styles of writing. Only the printed samples were selected for the data set. This resulted in a data set of 4 classes with 18 patterns in each class. This is not enough data to have meaningful results, so hand-printed samples of the words 'buffalo', 'city', 'washington', and 'vegas' were collected from various people in our lab. The final data set for the 4-class problem contained 400 samples, 100 from each class.

3.2.2 Examples of the Words. Figure 3.1 displays some examples of the data.

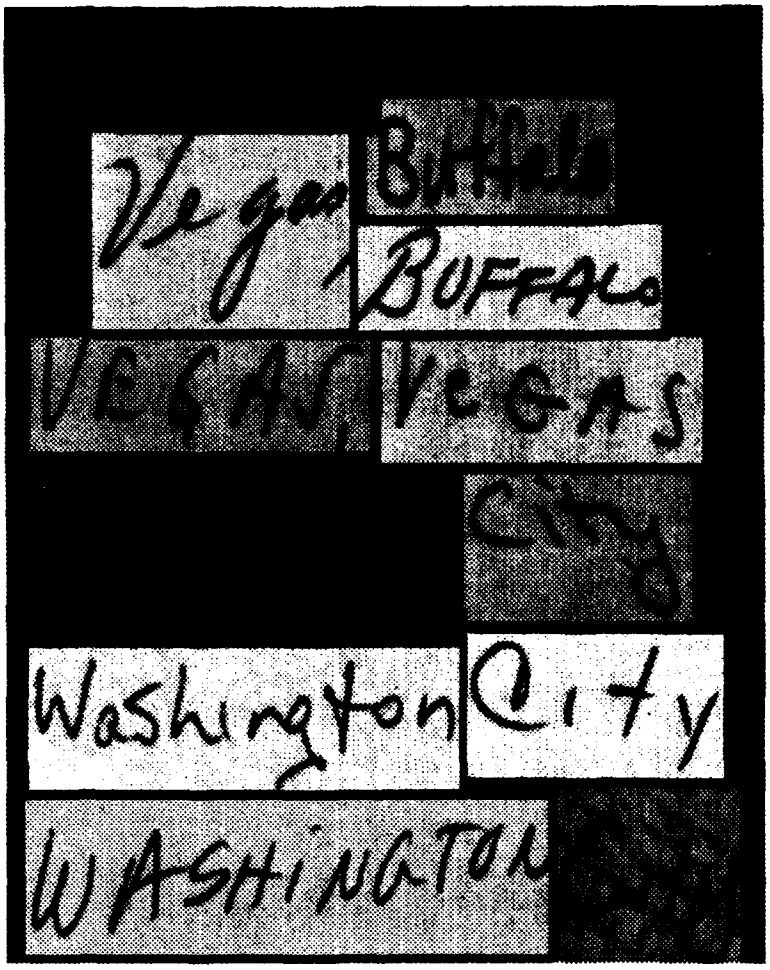

Figure 3.1 Examples of Handwritten Words

3.2.3 Searching the Database for Words. After determining which words will be used for an experiment, a script file to search the database for the particular words desired can be used. The script file searches the image truth files and when it finds a word it is searching for, it converts the file to a sunraster image file. See Appendix B.

3.3 Preprocessing the Images

Since the words were already well segmented, it is only necessary to do some minor preprocessing. The two preprocessing techniques used are described below.

3.3.1 Binarization. A binarization of the image is accomplished mainly to standardize the images. Some of the images have darker backgrounds than others. After binarization, a pixel value that represents part of the word has a pixel value of zero and any part of the image that is background has a pixel value of 255. Figure 3.2 illustrates the original image before binarization. 3.3 illustrates the image after binarization.

3.3.2 Window the image. In this step, the image is cropped so that the word completely fills the window of the image. This technique is done based on the advice of Dr. Kabrisky. Capt O'Hair did all his work using words that completely filled the window (17). Performing this step provides scale invariance. Figure 3.4 illustrates the cropped image. Again, this preprocessing method assumes that word segmentation can be accomplished.

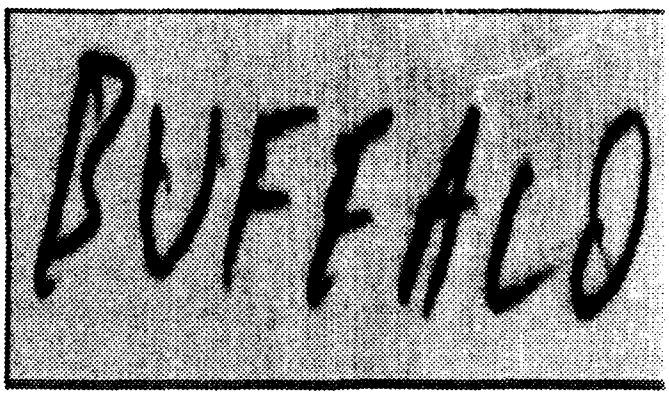

Figure 3.2 Original Image

Figure 3.3 Binarized Image

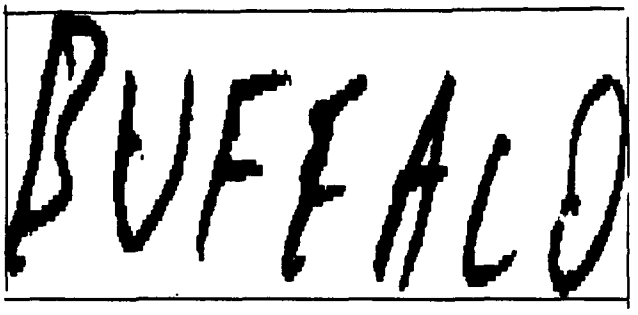

Figure 3.4 Cropped Image

3.4 Feature Extraction

3.4.1 Fourier Feature Extraction. The two-dimensional discrete Fourier Transform is described by the following equation:

$$S(f_x, f_y) = \sum_x \sum_y s(x,y)[cos2\pi(f_x x + f_y y) - isin2\pi(f_x x + f_y y)]$$

where,

- f_x = spatial frequency in x
- f_y = spatial frequency in y
- M = height of image in pixels
- N = length of image in pixels
- Range of spatial frequencies in x to calculate up to 10 harmonics: $-10/N, -9/N, ..., 9/N, 10/N$
- Range of spatial frequencies in y to calculate up to 10 harmonics: $-10/M, -9/M, ..., 9/M, 10/M$
- x,y = location of real valued input
- s(x,y) = intensity of image at location x,y

Each image of a handwritten word is in the form of a two-dimensional array of gray level pixel values ranging from 0 to 255. The intensity, s(x,y), of the image at pixel location x,y is a value 0 or 255. The array has a height M and a length N.

By taking advantage of symmetry of the Fourier Transform, only half of the coefficients need to be calculated. For example, the cosine is an even function and the sine is an odd function so the the following properties apply: (8)

- Re[F(A,B)] = Re[F(-A, -B)]
- Re[F(-A,B)] = Re[F(A, -B)]
- Im[F(A,B)] = -Im[F(-A, -B)]
- Im[F(-A,B)] = -Im[F(A, -B)]

When calculating discrete values of the Fourier Transform up to the third harmonic, a total of 49 cosine (48 plus dc term) and 49 sine terms (dc term is zero) are produced.

Using the properties listed above, only half the cosine and sine terms need to be calculated. This results in 25 cosine terms and 24 sine terms. Now 49 unique coefficients are calculated and no duplication exists. This is not only important to reduce calculations but it also eliminates redundancy in the feature set.

3.5 Energy Normalization

Once the Fourier coefficients are calculated, they are energy normalized. Since each image of a word has a different size pixel array, a means to normalize the values to an even playing ground is necessary. The energy normalization used in this case is to divide each term by the square-root of the sum of the squares of each coefficient. The following formula describes this.

$$< S_{r,c} >= \frac{S_{r,c}}{[\sum_{r=1}^{2n+1} \sum_{c=1}^{2n+1} S_{r,c}^2]^{1/2}} \tag{3.1}$$

where,

- $< S_{r,c} >$ = the normalized (r,c)'th element
- r = rows
- c = columns
- n = number of harmonics

3.6 Calculating 2D Discrete Fourier Transform

For each image, the Fourier Transform was computed. A total of 10 harmonics were computed. Effectively this implements a 21x21 spatial filter in the frequency domain. A total of 441 numbers result for each word image. From this set of numbers, a more specific spatial frequency filter can be extracted. For example, a 7×7 spatial frequency filter which captures the lower 3 harmonics, or a 3x5 spatial filter, etc.

Figure 3.5 illustrates the numbers used to specify a particular Fourier coefficient when computing the Fourier Transform. The features above the horizontal line are the cosine terms and the features below the line are the sine terms. The dc term is in the middle. Each square outline represents a particular harmonic. The results chapter refers

to these numbers. When a 7×7 spatial frequency filter is mentioned, it is referring to the 7×7 square centered on the dc term.

Figure 3.6 and 3.7 illustrate the reconstruction of a word image from the lower three harmonics and lower ten harmonics respectively.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126
127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147
148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168
169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189
190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231
232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252
253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273
274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294
295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315
316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336
337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357
358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378
379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420
421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441

Figure 3.5 Feature Numbers for 10 Harmonics of the 2-dimensional discrete Fourier Transform

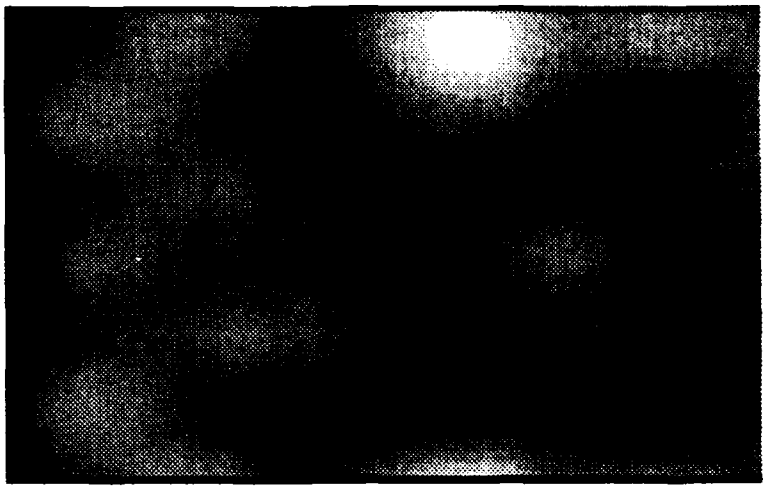

Figure 3.6 Image of 'Buffalo' Reconstructed From Three Harmonics

Figure 3.7 Image of 'Buffalo' Reconstructed From Ten Harmonics

3.6.1 Figure of Merit features. "The ability of a feature to separate two classes depends on the distance between classes and the scatter within classes (18)." Fisher's discriminant is a method to characterize the separability of features from two classes based on distance between classes and scatter within classes. This method is based on the following equation from Parsons (18).

$$f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{3.2}$$

An extension to multiple classes is defined by the F ratio equation or generalized Fisher's discriminant defined in Equation 3.3 (18).

$$F = \frac{\text{variance of the means(over all classes in one dim.)}}{\text{mean of the variance(within classes in one dim.)}} \tag{3.3}$$

For n features and m classes this is defined mathematically,

$$F = \frac{[1/(m-1)\sum_{j=1}^{m}(\mu_j - \bar{\mu})^2]}{[1/(m(n-1)\sum_{j=1}^{m}\sum_{i=1}^{n}(x_{ij} - \mu_j)^2)]} \tag{3.4}$$

where,

- x_{ij} = ith feature for class j
- μ_j = mean of all features for class j
- $\bar{\mu}$ = mean of all measurements over all classes

This can be used to evaluate each feature's ability to separate the classes. The higher the F ratio, the more separable it is and therefore more easily classifiable. For the 441 features that were calculated by the Fourier transform, an F ratio was calculated for each feature over 100 samples from each class. A rank ordering of the F ratio was done to see which features had the most separability. According to Parson's, selecting the best F ratio features for classification is not safe (18). Although, selecting the best group of features for classification can be heuristic in nature. This assumption will be tested.

3.7 Feature Subset Evaluation

Once all 441 features are calculated from the Fourier Transform, it is desirable to find a subset of the 441 features with which to use for classification. Not only will a subset reduce the amount of computations, but it also may reveal a much better feature set with which to classify. Several methods are used to try to select the right group of features.

The methods discussed are spatially filtering the Fourier Transform, calculating a figure of merit on each dimension of the data, calculating the Karhunen-Loeve transform, and using magnitude and phase information.

3.7.1 A 7×7 Spatial Filter. As outlined in Chapter 2 of this document, the lower 3 harmonics have proven very successful in terms of classification of digits and machine printed words. So naturally, this is one feature set that is used. The feature subset is then developed by extracting the 7×7 array of features from the 21x21 array calculated by the Fourier Transform. Once the subset is established, the classification is performed.

3.7.2 Figure-of-Merit. The figure of merit or F-ratio is calculated on all 441 dimensions of the feature set. The result is a rank ordering of F-ratios and their corresponding feature. Again, the higher the F-ratio, the more separable the data in that particular dimension. From this ranking, the top ten features are selected to be members of the feature subset. The classification technique performed is the addon technique described later.

In addition, the top 49 figure of merit features are selected for another feature subset. This will allow a comparison between using features from the lower 3 harmonics versus using the highest ranking figure-of-merit features.

3.7.3 Karhunen-Loeve Transformation. The goal in doing a Karhunen-Loeve transformation on the feature set is to obtain features which are best suited for separating classes (18). To compute the Karhunen-Loeve transform of the feature set, the covariance matrix of the feature set is computed, then the eigenvalues and respective eigenvectors of the covariance matrix are calculated and arranged in descending eigenvalue order (28). Dimensionality of the original feature set is reduced by the number of eigenvectors selected. The eigenvectors selected make up the transformation matrix A in the equation, $\mathbf{y} = \mathbf{A}\,\mathbf{x}$, where x is the original feature set and y is the new feature set (18). The eigenvalues give the variance of the new features in y (18). The new feature space contains a set of features with the greatest variance and thus these features are considered the "true" features (18). The features with low variances are considered noise and were eliminated (18). In effect,

the transformation has computed a new orthogonal feature space with each dimension corresponding to a variance, the first dimension having the greatest variance.

A Karhunen-Loeve transform was computed on the original feature set of 441 features calculated from the Fourier Transform. The 441 dimensions were reduced to 10 dimensions by placing 10 eigenvectors with the largest eigenvalue in the transformation matrix, A. A new feature subset consisting of 10 features per sample is then tested for classification.

3.7.4 Magnitude and Phase. Since so much information of an object is contained in the phase information, another feature subset was developed by computing the phase of a 7×7 spatial frequency filter. In addition, the magnitude was computed for another feature subset. The phase was obtained by computing the arctan of the sine term divided by the cosine term for a specific set of spatial frequencies. For example, 24 unique phase terms are calculated from the 49 unique terms generated from calculating 3 harmonics of the Fourier Transform. The magnitude is obtained by computing the square root of the sum of the squares of the specific cosine and sine term.

Four separate feature subsets were generated. The first feature subset consisted of 24 features per sample representing the phase of a 7×7 spatial frequency filter. The second feature subset contained 25 features per sample representing the magnitude. The dc term is also included. The third feature subset contained 49 features representing both the phase and the magnitude. The fourth feature subset contained only the imaginary components or the sine terms. This subset consisted of 24 features per sample.

3.7.5 Add-on Procedure. Ideally, it would be desirable to test all combinations of features to find the best set with which to classify; however, the amount of computation is prohibitive for this method, especially for a large number of features. There are other methods which try to pair down the best combinations of features without testing each combination. One method is called the add-on procedure (18).

In the add-on procedure, every feature from the feature set is evaluated by testing its classification ability. The best individual feature is then selected to be evaluated in combination with each remaining feature. The best combination of the two features is then

selected to be evaluated in combination with each remaining feature. The best combination of three features... and so on until the desired subset of features is attained. The desired subset of features is the one that gives the best performance. This method requires only k(2N + 1 -k)/2 evaluations, where k is the number of features in the subset and N is the total number of features under consideration (18). The 441 features calculated from the Fourier Transform is a large number to evaluate even using the add-on procedure. The feature subsets described above are the feature set that are classified using this addon procedure.

3.7.6 Miscellaneous Feature Subsets. A few additional feature subsets were generated for testing. In the case of the 2-class problem, each harmonic was evaluated individually, with and without the dc term. Further, for the 2-class problem, feature sets were built by adding single features at a time, from the lower three harmonics and from the FOM features, up to 49 features. Testing was completed after each addition of a feature.

3.8 Data Normalization

To achieve the feature set's invariance to scale and displacement, data normalization is done (5). This is accomplished by calculating the means and standard deviation for each input dimension. Then each input dimension, for each sample, is subtracted by the mean and divided by the standard deviation for that dimension. The end result is data with zero mean and unit variance in all dimensions.

It is interesting to note that the F ratio is the same whether or not the data has been normalized. This is because multiplying a random variable by a constant multiplies its variance by the square of that constant while adding a constant to a random variable leaves the variance unchanged (4). In the F ratio, the variance of the means will in effect by multiplied by $1/s^2$, where s is the variance and the means of the variance will also be multiplied by $1/s^2$. The $1/s^2$ will be cancelled out and the F ratio remains unchanged.

3.9 Classification

The classification tools used in this study are the multi-layer perceptron and the knn classifier as described in Chapter 2 of this document. A program specifically designed for pattern recognition, called LNKnet (12), was used for classification using the multi-layer perceptron and the knn classifier. Some modification of the program is necessary to employ the add-on technique and to test the data set after each epoch of training. The appendices contains the script files that are used to accomplish this.

3.10 LNKnet Parameters

Every test conducted in LNKnet, using the multi-layer perceptron, had the same parameters. A list of the parameters is given below.

- Training set: 50 random samples/class
- Test set: 50 random samples/class
- Epoch of training: 25
- Number of hidden nodes: 50
- Step size : 0.05
- Momentum : 0.0
- Tolerance : 0.2 (if the error of an output node is less than 0.2, the weights are not updated)
- Decay 0.0
- Error Function : Squared Error
- Output Node Function : Standard Sigmoid
- Weight update after each trial
- Random presentation order

3.11 Conclusion

This chapter described the methods used to determine whether or not Fourier coefficients computed from the whole word are of value in classifying hand-printed words. A description of the data set used and how the data was preprocessed was given. Then, computing the Fourier Transform of each word image resulted in a 441 dimensional feature vector for each word. All the feature vectors together built the feature set. From this feature set, many feature subsets were selected. The feature subsets consisted of figure-of-merit features, 7×7 spatial filter features, combinations of magnitude and phase features, Karhunen-Loeve features, and features from add-on testing. The next chapter reports the results of 2-class and 4-class testing using these feature sets.

IV. Results

4.1 Results of a 2-class problem using figure-of-merit features

To determine whether using Fourier coefficients as features for recognition is worthwhile, a simple 2-class problem was attempted. The two classes for this problem are the words 'Buffalo' and 'City.' Beginning with the pixelized image of the word and computing the discrete Fourier transform of the image, the resulting coefficients, both cosine and sine terms, form the feature set. Fourier coefficients up to the tenth harmonic are calculated. It is from this feature set that the best subset of features are sought. In this case, figure-of-merit features are used.

A figure-of-merit was computed across each dimension. In other words, the separability of each dimension was calculated independent of any other dimension. A number is given corresponding to it separability. The higher the number, the better separation between classes for that dimension. Selecting the features with the highest figure-of-merit is how the feature subset is built.

The "add-on" procedure described in Chapter 3 is the method used to find the best combination of 5 features out the the ten features with the highest figure-of-merit. So actually, a subset of features is obtained from the ten features with the highest figure-of-merit. Then, from this subset, the best combination of 5 features forms the final feature subset.

Table 4.1 lists the top ten figure-of-merit features, their corresponding figure-of-merit and harmonic. By referring to Figure 3-5, the feature number listed can be traced to the particular Fourier coefficient calculated. One of the reasons why ten harmonics is calculated is to find out if any of the higher harmonics have good separability. It is interesting to note that all but one of the highest F-ratios are within a 7×7 low pass spatial frequency filter. Considering the previous success with using 7×7 spatial filters on recognition of characters, it is not surprising to find that the most separable dimensions are within the 3 lowest harmonics. To give some meaning to the figure-of-merit numbers listed in table 4.1, Figure 4.2 plots a histogram of feature number 245 for 100 samples from both classes. This illustrates the meaning of the figure-of-merit, .32, for feature number 245. Observing

Table 4.1 Top 10 Most Separable Features, Their Corresponding FOM and Harmonic

Feature	Figure-of-Merit	Harmonic
245	0.32	3
222	0.25	1
242	0.23	1
178	0.22	2
239	0.21	3
179	0.18	2
216	0.18	5
220	0.17	1
180	0.16	2
198	0.13	2

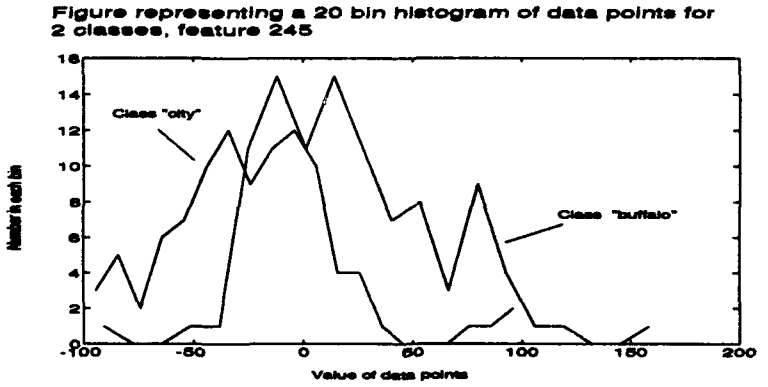

Figure 4.1 Histogram of Feature 198 for 200 Samples, 100/class

this graph, it shows that the feature with the highest figure-of-merit is not very separable. For comparison, 4.7 shows the histogram for feature 198 with a 0.13 figure-of-merit.

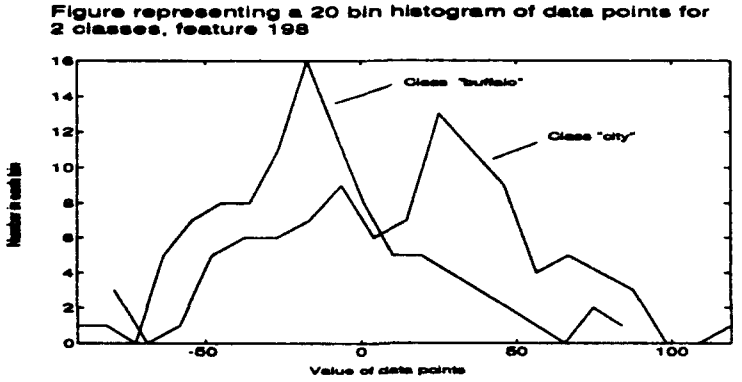

Figure 4.2 Histogram of Feature 198 for 200 Samples, 100/class

Since all remaining features have a lower F-ratio than feature number 245, no single feature alone is usable for classification; however, the combination of two or more not very separable features may further separate the classes. The "addon" procedure will find a combination of features that will increase the separability.

The training set and test set both consisted of 100 samples randomly selected and evenly distributed for each class. Three trials were performed and the results were averaged. The training error reported corresponds to the lowest test error. So when the lowest test error occurred, the corresponding training error was noted. When training the multilayer perceptron it is possible that as the training error decreases and eventually reaches zero, the test error may not be the lowest. Figure 4.3 illustrates this effect during a run. The test set was tested after each epoch of training. As the figure shows, the test error decreases along with the training error but it reaches a point, at 23% error after 11 epochs, where it begins to increase even though the training error is decreasing. To overcome this effect, testing was conducted after each epoch of training and the lowest test error was selected. Tables 4.2 and 4.3 shows the results of classification testing using the addon procedure to find the best combination of 5 features out of the 10 FOM features. The test error reported indicate the lowest test error during training and the training error reported is where the test error was minimized. The best combination of 5 features did not constitute the top

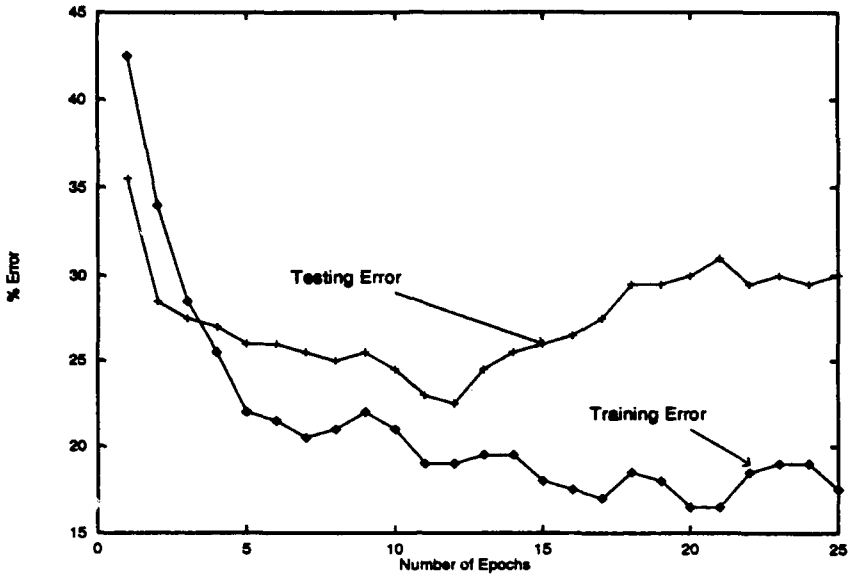

Figure 4.3 Training and Testing Error vs Epochs of Training

5 FOM features. This supports the notion that the combination of 2 relatively separable features may not be the best combination of any 2 features. After the combination is made the separability must be determined. In this case, the classification method determined the separability of the combination of features by the test results. Using this method, the multilayer perceptron performed the best. 84 out of 100 were correctly classified using 5 features.

Table 4.2 Best Combinations of Top 10 FOM Features and the Resulting MLP Test Error

Feature	% Error(testing)	Rmserr(testing)	% Error(training)
222	26	.44	35
222,178	21	.42	26
222,178,180	22	.43	25
222,178,180,245	19	.41	21
222,178,180,245,198	16	.38	17

4.2 Results of 2-class Problem Using 49 Features From Lower Three Harmonics

A test was done to evaluate the test error per addition of single features, beginning with one feature and adding a feature up to 49 features. In this case the features were the

Table 4.3 Best Combinations of Top 10 FOM Features and the Resulting 1-nn Test Error

Feature	% Error(testing)	Rmserr(testing)
178	40	.63
178,179	36	.6
178,179,180	30	.55
178,179,180,222	28	.53
178,179,180,222,198	26	.5

49 features of the 3 lower harmonics. Notice the test error drops quickly as a few features are added then it reaches a point where no improvement in test error is achieved with additional features. The lowest test error was 19.0%. Not only is it important to find good features, but it is just as important to find the right number of features that will give the best recognition. Figure 4.2 shows the results and Table 4.4 lists the specific features.

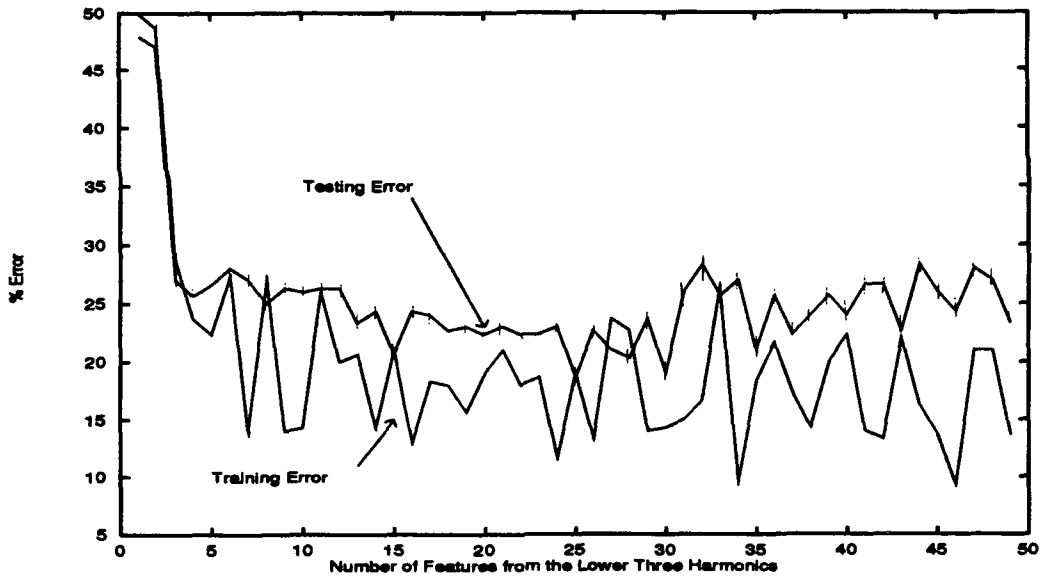

Figure 4.4 MLP Test Error vs Number of Features in the Lower Three Harmonics

4.3 Results of 2-class Problem Using 49 FOM features

The same test described in the previous section was completed using 49 FOM features. In this case, the FOM features used were the ones with the highest FOM. The first feature corresponded to the highest figure of merit and each additional feature cor-

Table 4.4 Listing of the 49 Fourier Features Used

Ftr	FT coeff	Harmonic	Ftr	FT coeff	Harmonic	Ftr	FT coeff	Harmonic
1	221	dc	10	177	2nd	26	155	3rd
2	222	1st	11	261		27	281	
3	220		12	178		28	156	
4	199		13	262		29	282	
5	243		14	179		30	157	
6	201		15	263		31	283	
7	241		16	180		32	158	
8	200		17	264		33	284	
9	242		18	181		34	159	
			19	265		35	285	
			20	198		36	160	
			21	240		37	286	
			22	202		38	161	
			23	244		39	287	
			24	219		40	176	
			25	223		41	260	
						42	197	
						43	239	
						44	182	
						45	266	
						46	203	
						47	245	
						48	218	
						49	224	

responded to the next highest figure of merit. Again, the test error drops quickly but reaches a point where additional features do not enhance recognition. Figures 4.5 and 4.6 shows the results of classification testing and table 4.5 lists the specific features. Notice the test error when classifying with the 1-nn, that is, the test error reaches a minimum and then slowly increases with additional features. This supports Dr. Kabrisky's theory that the addition of more features only adds noise and hence, the recognition performance is decreased.

Table 4.5 List of the 49 FOM features

Num	FOM Feature												
1	245	8	220	15	285	22	260	29	207	36	135	43	165
2	222	9	180	16	201	23	163	30	247	37	219	44	224
3	242	10	198	17	223	24	261	31	158	38	197	45	119
4	178	11	243	18	246	25	269	32	174	39	240	46	230
5	239	12	264	19	244	26	193	33	182	40	241	47	199
6	179	13	218	20	196	27	370	34	227	41	327	48	268
7	216	14	203	21	237	28	226	35	234	42	229	49	342

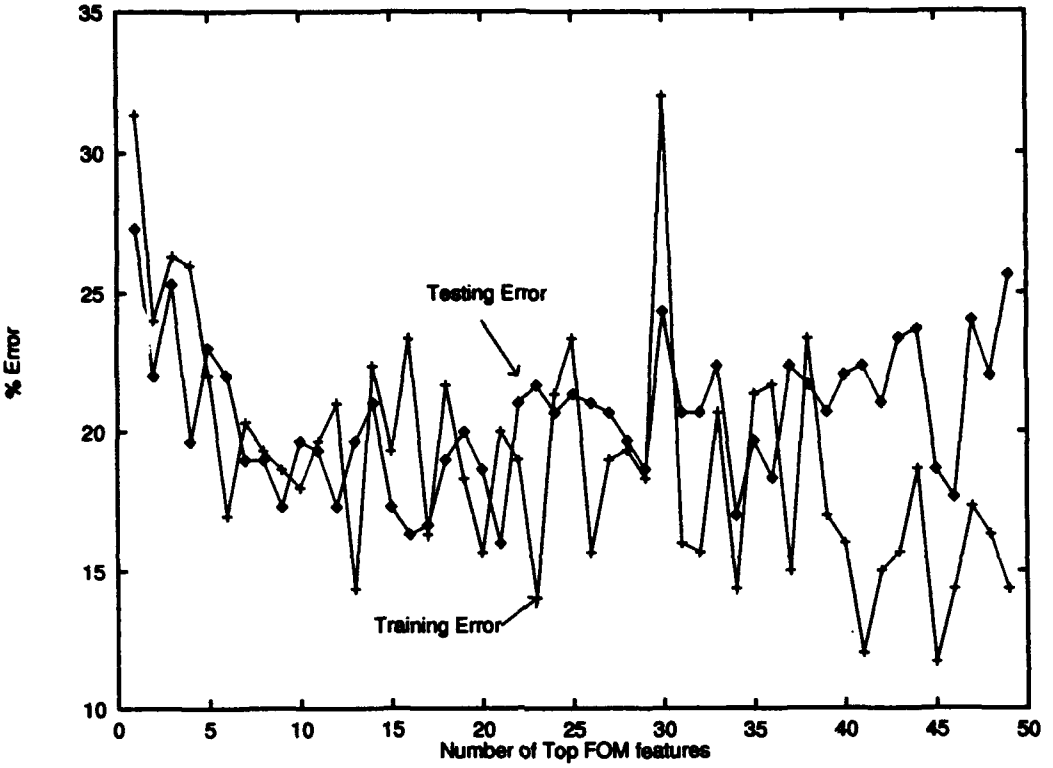

Figure 4.5 MLP Test Error vs Number of Top FOM Features

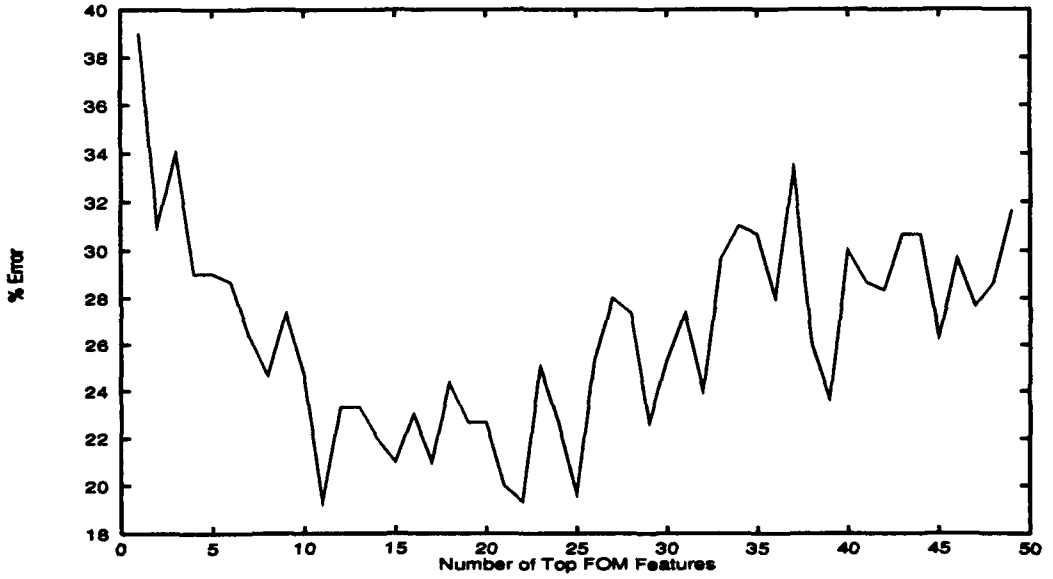

Figure 4.6 1-nn test error vs number of top FOM features

4.4 Results of Some Miscellaneous Feature Subsets

To conclude the testing of the 2-class problem, some miscellaneous feature subsets were built and tested. Table 4.6 lists the results. The imaginary terms alone performed well for recognition purposes as did the first harmonic. The performance was degraded for each subsequent harmonic.

Table 4.6 MLP Test Results from Various Feature Combinations

Features	% Error(testing)	Rmserr(testing)	% Error(training)
3 x 5	23	.32	20
imag only(lower three harmonics)	17	.31	14
imag only(lower three harmonics) plus dc term	25	.35	16
1st harmonic	23	.31	27
1st harmonic plus dc	24	.34	18
2nd harmonic	26	.36	21
2nd harmonic plus dc	22	.34	25
3rd harmonic	30	.39	15
3rd harmonic plus dc	30	.38	13

4.5 4-Class Results

The results of the 2-class problem show promise in using Fourier coefficients as features. Further testing, using more feature subsets, and additional classes are needed to confirm the recognition capability of Fourier coefficients. The next set of results examines the use of several new feature subsets. Two of the same sets used for the two class problem are again used for the 4-class problem. They include the 7×7 Spatial Frequency Filter and the 10 figure-of-merit features. A more in-depth look at feature subsets is completed by the addition of magnitude, phase, and Karhunen-Loeve features. The two additional classes include 'Vegas' and 'Washington.'

4.6 Results of a 4-class Problem Using FOM features

This experiment is the same as described in the 2-class problem using figure-of-merit features. Table 4.7 lists the top ten figure-of-merit features, their corresponding figure-of-merit and harmonic for the 4-class data. By referring to figure 3-1, the feature number listed can be traced to the particular Fourier coefficient calculated. Again, it is interesting

Table 4.7 Top 10 Most Separable Features, Their Corresponding FOM and Harmonic

Feature	Figure-of-Merit	Harmonic
198	0.68	1
243	0.62	1
200	0.53	1
179	0.49	2
219	0.47	2
220	0.47	1
223	0.37	2
245	0.27	1
244	0.25	1
217	0.23	4

to note that all but one of the highest F-ratios are within a 7×7 low pass spatial frequency filter. In this case, six of the top ten are within the first harmonic. Not only is the bulk of the information in the lower harmonics, but also, this information is the most separable. Tables 4.8 and 4.9 shows the results of classification testing using the addon procedure to find the best combination of 5 features out of the 10 FOM features. The combination of five features correctly classified 73 out of 100.

Table 4.8 Best Combinations of Top 10 FOM Features and the Resulting MLP Test Error

Feature	% Error(testing)	Rmserr(testing)	% Error(training)
198	56	.43	58
198,223	45	.40	49
198,223,219	36	.37	38
198,223,219,243	32	.33	34
198,223,219,243,200	27	.30	24

Table 4.9 Confusion matrix for Best Combination of 5 FOM features

Actual class	Classified			
	0	1	2	3
0	35	3	7	5
1	0	31	19	0
2	0	4	44	2
3	6	1	3	40

4.7 Results of a 4 class Problem Using Features from a 7×7 Low-pass Spatial Frequency Filter

This section reports on the results of using a 7×7 spatial frequency filter on the Fourier coefficients. The feature set consisted of 49 terms which included 24 cosine and sine terms plus the dc term. Classification testing was completed using both the multilayer perceptron and the knn classifier. For the multilayer perceptron, only one parameter was changed and that was the number of hidden nodes. A test was done with 50, 100, and 200 hidden nodes to see if the number of hidden nodes had any effect. For the knn classifier the only parameter changed was the number of k neighbors. Four different neighbors were used, they were 1, 3, 5, and 7.

Table 4.10 lists the results of this set of testing by the multilayer perceptron. The number of hidden nodes in the multilayer perceptron had little effect. By increasing the number of classes, the recognition performance decreased to 62% correct from the 77% percent achieved in the 2-class case. This indicates that the features tend to cluster together with the addition of more classes. Table 4.11 list the results of the same set using the k-nn classifier. The knn classification rate of 26.5% on the test set surpassed the results of testing with the multilayer perceptron. Calculating the distance to additional neighbors actually degrades the performance of the knn classifier. Table 4.12 list the confusion matrix on the test set with the best classification rate for this section of testing. The references to the classes are as follows: 0 - 'Buffalo', 1 - 'Vegas', 2 - 'Washington', 3 - 'City'. The recognition rate for each class is similar, not one class was easily recognizable. This particular set of features have discrimination potential. The results indicate that as a rough cut for classifying words, a 73.5% recognition rate was achieved. Clearly, differences in handwritten words are seen in the lower 3 harmonics of the Fourier Transform. Finally,

Table 4.10 MLP Test Results Using 49 Features From the 7x7 Low-pass Spatial Filter

Hidden Nodes	% Error(testing)	Rmserr(testing)	% Error(training)
50	39	.37	12
100	38	.39	8
200	40	.42	9

Figure 4.7 illustrates the test error as individual features are added one at a time.

Table 4.11 K-nn Test Results Using 49 Features From the 7×7 Low-pass Spatial Filter

k	% Error (testing)	Rmserr (testing)
1	26.5	.364
3	27.0	.315
5	32.5	.323
7	35.0	.339

Table 4.12 Confusion matrix for the 1-nn Using 49 Feature From the 7×7 Low-pass Spatial Filter

Actual class	Classified			
	0	1	2	3
0	34	2	7	7
1	2	35	13	0
2	1	11	37	1
3	9	0	0	41

4.8 Results of a 4 class Problem Using Features from a 7×7 Low-pass Spatial Frequency Filter in add-on testing

This section reports the results of using add-on testing of the 49 Fourier coefficients from the 7×7 spatial frequency filter. A recognition rate of 68.7% was achieved. This compares favorably to the results of the figure-of-merit features because three of the features of the best combination are the same. It is interesting to note that the first feature, 198, was the best feature in both figure-of-merit and this test. Feature 198 tested better with another feature that was not in the top 10 figure of merit features; however, the subsequent combinations returned a lower test error. This supports using figure-of-merit vs straight add-on. Table 4.13 lists the results.

4.9 Results of a 4-class Problem Using Magnitude and Phase Features

The next set of features used for classification were based on the magnitude and phase of the Fourier Transform. Specifically, the magnitude and phase of the coefficients resulting from using a 7×7 spatial frequency filter. In addition, a feature set of only the imaginary components resulting from using a 7×7 spatial frequency filter was tested.

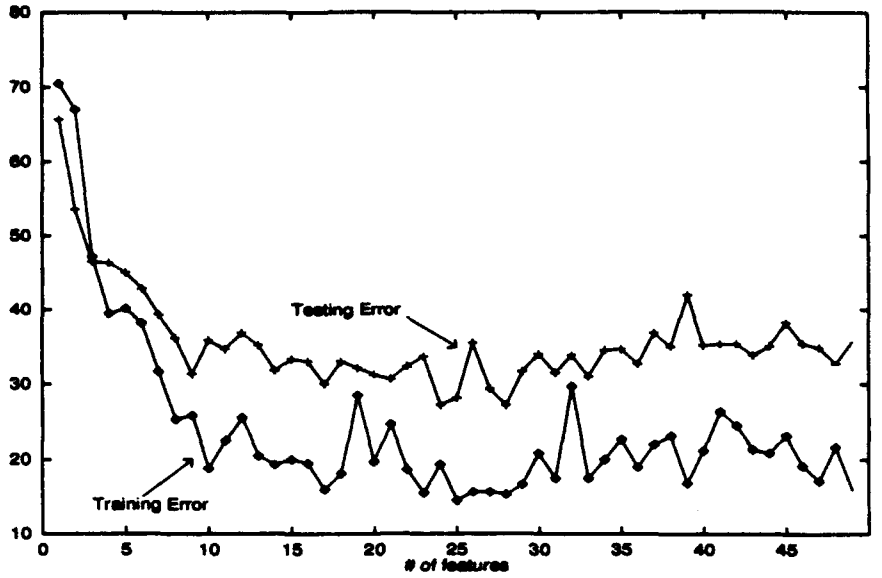

Figure 4.7 MLP Test Error vs Number of Features From The Lower 3 Harmonics

Table 4.13 Best combinations of 49 features from the 7×7 spatial frequency filter and the resulting MLP test error

Feature	% Error(testing)	Rmserr(testing)	% Error(training)
198	54.5	.48	50.2
198,260	44.5	.45	41
198,260,241	38	.39	42
198,260,241,243	34.5	.36	25.5
198,260,241,243,223	31.3	.35	22

4.9.1 Results of a 4-class Problem Using Magnitude and Phase. This particular set of features included both the magnitude terms and the phase terms. A total of 49 features were used for this experiment. Again, for the multilayer perceptron, only one parameter was changed and that was the number of hidden nodes. A test was done with 50, 100, and 200 hidden nodes to see if the number of hidden nodes had any effect. For the knn classifier the only parameter changed was the number of k neighbors. Four different neighbors were used, they were 1, 3, 5, and 7.

Table 4.14 and Table 4.15 show the results of this group of classification testing using the multilayer perceptron. The best error rate for this set of features was 46% for 100 hidden nodes using the multilayer perceptron and 41.5% using the 1-nn. Table 4.16

and Table 4.17 show the results of this group of classification testing. Each confusion matrix corresponds to the best recognition rate.

Table 4.14 MLP test results using 49 features from the magnitude and phase of the 7×7 low-pass spatial filter

Hidden Nodes	% Error(testing)	Rmserr(testing)	% Error(training)
50	50	.41	22
100	46	.4	21
200	47	.44	12

Table 4.15 Confusion matrix for magnitude and phase using 100 hidden nodes in the multilayer perceptron

Actual class	Classified			
	0	1	2	3
0	24	5	1	20
1	2	35	6	7
2	3	26	17	4
3	11	6	2	31

Table 4.16 K-nn test results using magnitude and phase features from the coefficients resulting from a 7×7 low-pass spatial frequency filter

k	% Error (testing)	Rmserr (testing)
1	41.5	.456
3	45.5	.395
5	46.5	.374
7	42.5	.369

4.9.2 Results of 4-class Problem Using Phase Features Only. The next set of features were based only on the phase information of the 7×7 low pass spatial frequency filter. The feature set consisted of 24 terms per sample. The knn performed better than the multilayer perceptron. The top recognition rate of 62.0% occurred with 7 nearest neighbors. This is the first experiment that shows a better recognition rate for greater than 1 nearest neighbors, in this case 7 nearest neighbors achieved the lowest test error.

Table 4.17 Confusion matrix for magnitude and phase using a 1-nn classifier

Actual class	Classified			
	0	1	2	3
0	26	6	3	15
1	3	36	8	3
2	4	16	28	2
3	17	1	5	27

Tables 4.18 and 4.20 show the results of each classifier testing. Tables 4.19 and 4.21 lists the confusion matrices for the best recognition rates. The phase contains most of the information, yet it did not support very separable features.

Table 4.18 MLP Test Results Using 24 Features From the Phase of the 7×7 Low-pass Spatial Filter

Hidden Nodes	% Error(testing)	Rmserr(testing)	% Error(training)
50	71	.44	58
100	60	.43	50
200	58	.44	40

Table 4.19 Confusion matrix for mlp with 200 using phase only features

Actual class	Classified			
	0	1	2	3
0	36	1	7	6
1	23	15	10	2
2	16	9	21	4
3	23	6	10	11

Table 4.20 K-nn Test Results Using Phase Features From the Coefficients Resulting From a 7×7 Low-pass Spatial Frequency Filter

k	% Error (testing)	Rmserr (testing)
1	42.5	.461
3	39.5	.377
5	40.5	.357
7	38.0	.354

Table 4.21 Confusion matrix for 7-nn using phase phase only features

Actual class	Classified			
	0	1	2	3
0	45	0	1	4
1	6	24	16	4
2	9	10	28	3
3	16	5	2	27

4.9.3 Results of 4-class Problem Using Magnitude Features Only. The next set of features were based only on the magnitude information. The knn performed much better than the multilayer perceptron. The top recognition rate of 59.5% occurred with 1 nearest neighbor. Both the multilayer perceptron and knn classified achieved similar classification results. Tables 4.22 and 4.24 show the results of each classifier testing. Tables 4.23 and 4.25 lists the confusion matrices for the best recognition rates.

Table 4.22 MLP test results using 25 features from the magnitude of the 7×7 low-pass spatial frequency filter

Hidden Nodes	% Error(testing)	Rmserr(testing)	% Error(training)
50	55	.42	19
100	42	.40	14
200	43	.42	12

Table 4.23 Confusion matrix for magnitude only using 100 hidden nodes in the mlp

Actual class	Classified			
	0	1	2	3
0	30	3	2	15
1	6	23	15	6
2	7	11	28	4
3	9	3	3	35

Table 4.24 K-nn test results using magnitude features from the coefficients resulting from a 7×7 low-pass spatial frequency filter

k	% Error (testing)	Rmserr (testing)
1	40.5	.466
3	44.5	.381
5	40.5	.380
7	46.5.0	.388

Table 4.25 Confusion matrix for magnitude features using a 5-nn classifier

Actual class	Classified			
	0	1	2	3
0	26	6	3	15
1	3	36	8	3
2	4	16	28	2
3	17	1	5	27

4.10 Results of 4-class Problem Using Imaginary Components Only

The next set of features were based only on the phase information. The knn performed much better than the multilayer perceptron. The top recognition rate of 59% occurred with 200 hidden nodes in the multilayer perceptron. Again, both the multilayer perceptron and knn classified achieved similar classification results. The results are also similar to the results of the phase only features. Tables 4.26 and 4.29 show the results of each classifier testing. Tables 4.27 and 4.28 lists the confusion matrices for the best recognition rates.

Table 4.26 MLP Test Results Using 24 Features From the Imaginary Components of the 7×7 Low-pass Spatial Filter

Hidden Nodes	% Error(testing)	Rmserr(testing)	% Error(training)
50	42	.39	10
100	43	.41	9
200	41	.42	10

Table 4.27 Confusion matrix for imaginary components only using mlp

Actual class	Classified			
	0	1	2	3
0	38	1	2	9
1	7	25	13	5
2	8	11	25	6
3	13	2	5	30

Table 4.28 K-nn Test Results Using Imaginary Components From the Coefficients Resulting From a 7×7 Low-pass Spatial Frequency Filter

k	% Error (testing)	Rmserr (testing)
1	44.0	.469
3	45.5	.413
5	44.0	.389
7	45.5	.393

Table 4.29 Confusion matrix for imaginary features using a 1-nn classifier

Actual class	Classified			
	0	1	2	3
0	25	2	17	6
1	2	22	25	1
2	1	7	41	1
3	17	2	7	24

4.11 Results of a 4-class Problem Using Combination of KLT Features

The final feature set developed was based on the Karhunen-Loeve Transform. In this test, ten features resulting from KL transforming the original 441 Fourier features were used in an add-on procedure. The best combination of five was kept as the feature set. The results of this testing show that the features that made up the best combination were the top five coefficients corresponding to the largest eigenvalues. These features represent the orthogonal directions in the feature space with the most variance. Tables 4.30 and 4.31 show the results of each classifier testing. Tables 4.32 and 4.33 lists the confusion matrices for the best recognition rates. The average recognition rate of 76.2% over three trails for the multilayer perceptron is the best rate for any feature subset selected. By performing the Karhunen-Loeve Transform on a set of correlated data, generated by calculating the Fourier Transform, the new KL components are now uncorrelated (18). This improves recognition performance. Figure 4.8 plots each data sample when the Karhunen-Loeve

Table 4.30 Best Combinations of Top 10 KLT Features and the Resulting MLP Test Error

Feature	% Error(testing)	Rmserr(testing)	% Error(training)
1	53	.46	64.8
1,2	45	.38	53.5
1,2,4	34.5	.35	42.3
1,2,4,5	28	.29	23.5
1,2,4,5,3	23.8	.28	19.33

transformation reduced the feature space to 2 dimensions. The patterns are well mixed together for two dimensions and the resulting test error was 45%.

Table 4.31 K-nn Test Results Using 5 KLT Coefficients

k	% Error (testing)	Rmserr (testing)
1	33.5	.409
3	30.0	.329
5	26.0	.311
7	24.0	.301

Table 4.32 Confusion matrix for best combination of 5 KLT features using mlp (76.2% accuracy)

Actual class	Classified			
	0	1	2	3
0	34	1	4	11
1	2	34	11	3
2	0	3	47	0
3	12	0	1	37

Table 4.33 Confusion matrix for best combination of 5 KLT features Using 7-nn classifier (76% accuracy)

Actual class	Classified			
	0	1	2	3
0	41	4	4	1
1	1	34	15	0
2	1	2	47	0
3	15	1	4	30

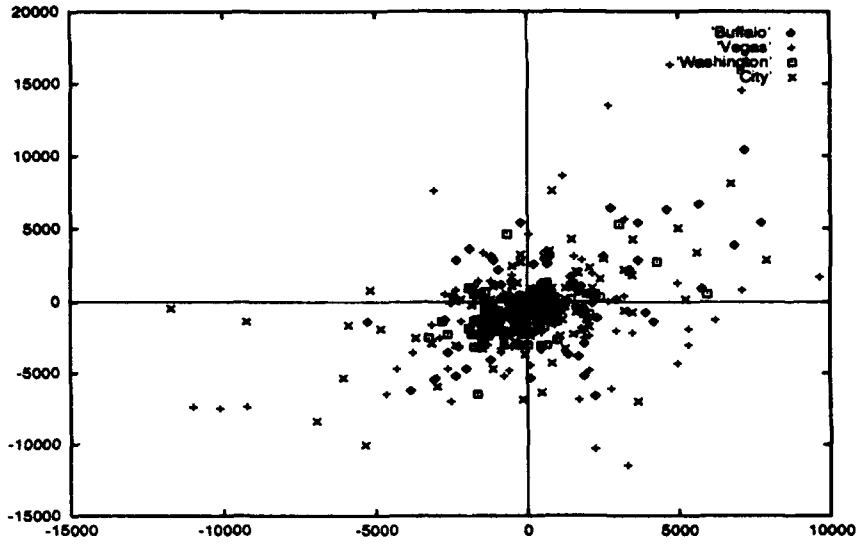

Figure 4.8 Patterns plotted for 2 dimensions of the KL transform

4.12 Images

Figure 4.9 shows a collection of commonly misclassified patterns. Many of the patterns have some type of stray lines in the image or the word itself has characteristics of cursive script. A step in analyzing why some patterns were correctly classified and why some were not, may be to look at the reconstruction of the images with the features used to classify them. In the following figures, two classes were analyzed. For each class, a correctly classified pattern and a misclassified pattern were reconstructed from the lower 3 harmonics to see if any noticeable difference could be seen. It does not appear from just a few images that there is any distinguishable difference. However, good reconstruction of the image does not mean those features will be good for recognition.

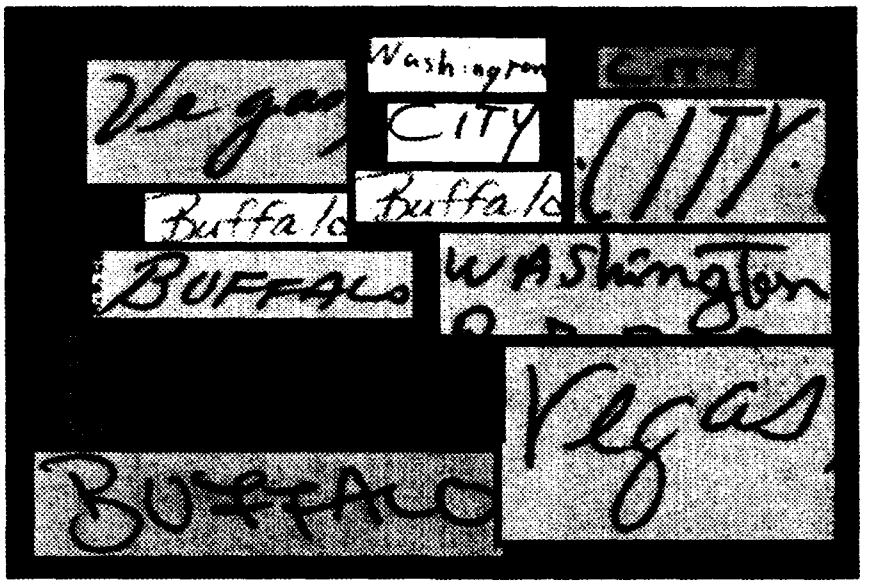

Figure 4.9 Mis-classified Patterns

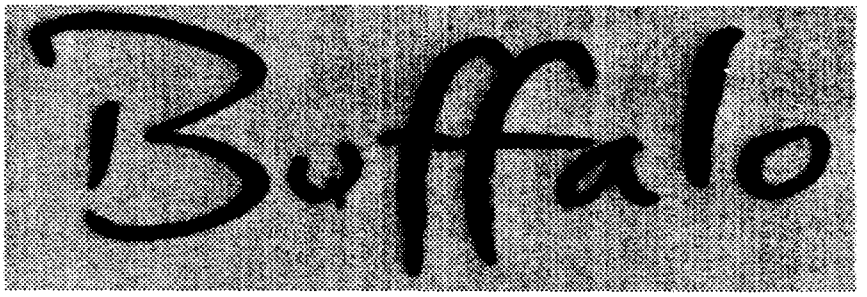

Figure 4.10 Correctly classified pattern in class 0

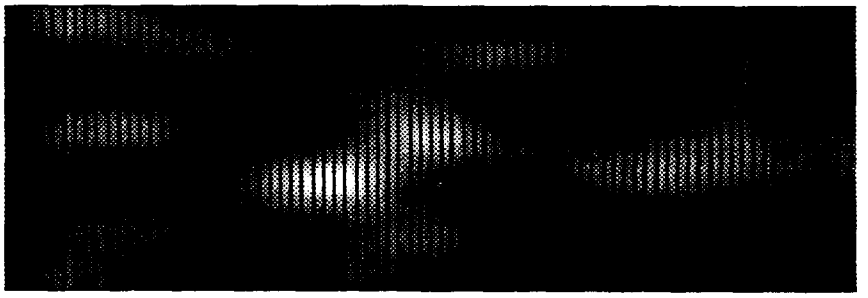

Figure 4.11 Correctly classified pattern in class 0, reconstructed using the lower three harmonics

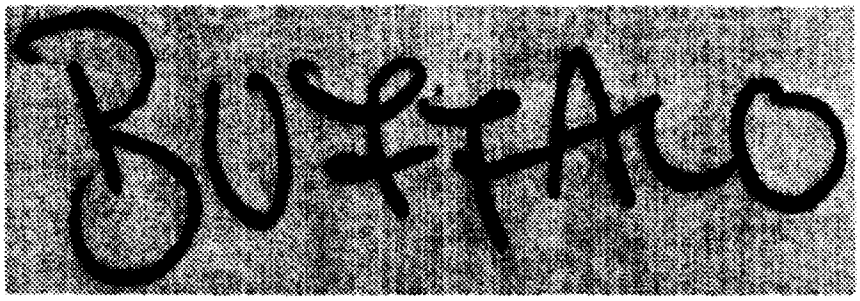

Figure 4.12 Mis-classified pattern in class 0

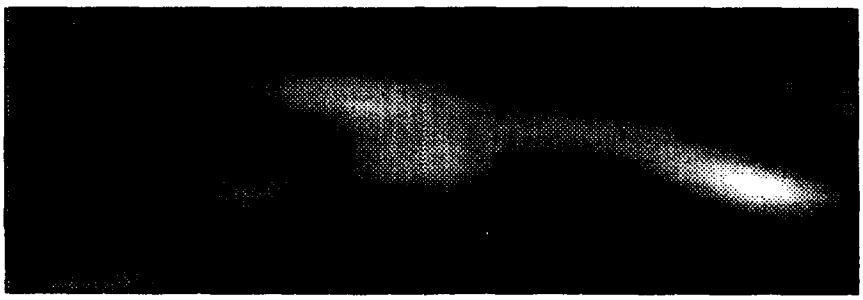

Figure 4.13 Incorrectly classified pattern in class 0, reconstructed using the lower three harmonics

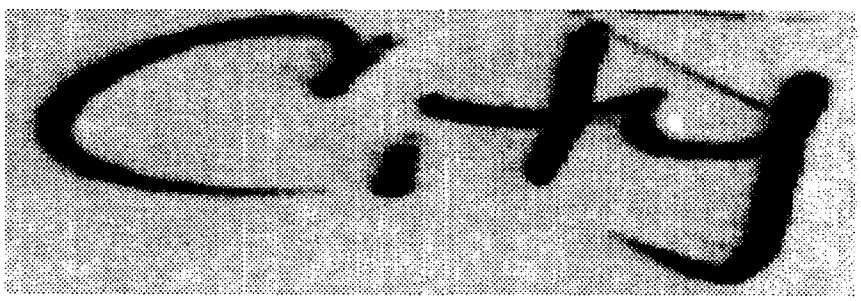

Figure 4.14 Correctly classified pattern in class 3

Figure 4.15 Correctly classified pattern in class 3, reconstructed using the lower three harmonics

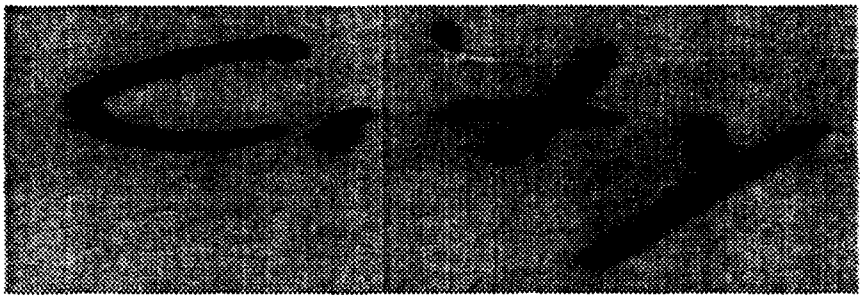

Figure 4.16 Mis-classified pattern in class 3

Figure 4.17 Incorrectly classified pattern in class 3, reconstructed using the lower three harmonics

4.13 Generalization of Recognition

What confidence can be placed on the results obtained? A good rule of thumb for the design of a pattern recognition system is to use half or less of the data for training (6). All the result obtained, used exactly half of the data for training and half for testing. Another standard for the design of the system is to use on the order of ten times c_k, where $c_k = 2(K+1)$ and K = the number of features (30). The best results were achieved using 5 KLT Fourier features, so in this case c_k equals 12. This would require 120 patterns per class for training. Only 50 patterns per class were used for training the system. Ideally, more patterns are required for better generalization. The true error rate is estimated to be in the range of 0.22 to 0.34 (19).

4.14 Conclusion

This chapter reported the results of testing several feature sets on both a 2-class and 4-class problem. Table 4.34 summarizes the results of each feature set used for the 4-class problem.

Table 4.34 Summary of 4-class testing

Feature Set	Recognition Rate (%)	Classification
7×7 Spatial Filter	74.5	1-nn
Figure-of-Merit	73	mlp
Magnitude and Phase	58.5	1-nn
Magnitude Only	59.5	1-nn
Phase Only	62	7-nn
Imaginary coefficients	59	mlp
Karhunen-Loeve	76.2	mlp

The 76.2% recognition rate using the Karhunen-Loeve transform of Fourier features was the top rate. Three of the feature sets resulted in recognition rates of greater than 70%. Good recognition was achieved for figure-of-merit and Karhunen-Loeve features using only 5 features. The high recognition rates indicate that Fourier Transform features are valuable in recognizing handwritten words.

V. Conclusions

The purpose of this thesis was to examine the use of Fourier Transform coefficients for the recognition of handwritten words. The pattern recognition problem consisted of classifying four handwritten words, 'Buffalo', 'Vegas', 'Washington', 'City.' Based on the success of using Fourier coefficients in the past for recognizing handwritten letters and machine-typed words, the logical next step was to analyze the recognition capability of Fourier coefficients of handwritten words. The analysis concentrated on searching for subsets of features from the Fourier coefficients computed of the word images. Several feature sets were generated to include using the coefficients from the 7x7 spatial frequency filter, figure-of-merit features, magnitude and phase, and Karhunen-Loeve features. Two methods of classification were used, the multi-layer perceptron and the k-nearest neighbor. LNKnet software provided the classification support and Khoros (14) provided image processing support.

This effort was a first cut at the problem of recognizing handwritten words. A more in-depth look at the Fourier coefficients was provided, such as the separability of each individual feature and the role the magnitude and phase played. The figure-of-merit feature set resulted in a 73% recognition rate. Using the magnitude and phase features together resulted in a 58.5% recognition rate, while magnitude features alone resulted in 59.5% and phase alone resulted in 62% recognition. Using the standard lower 3 harmonics, which had great success separating handwritten letters, produced a 74.5% recognition rate. In addition, applying the Karhunen-Loeve transform to further search for features with separability was examined. This resulted in the top performance of 76.2%. Other methods of feature selection included the addon procedure used during classification. All theses techniques were used in the attempt to pair down the infinite amount of feature combination to find the best set of features for classification. The result was a sufficient examination of Fourier coefficients used alone as features.

The results indicate that Fourier coefficients are beneficial to some degree in being able to classify handwritten words. The best recognition performance of 76.2% was achieved when the Karhunen-Loeve transform was computed on the Fourier coefficients. The variability in the handwritten word is difficult for the Fourier coefficients to overcome.

Although some recognition ability does remain, recognizing handwritten words requires more than just feature sets based on Fourier coefficients.

This leads to recommendations for future research in classifying handwritten words. The first recommendation is to incorporate a feedback mechanism into the classification process. The Fourier coefficients provide a good initial feature set for classification and can achieve 76.2% recognition, but more information is required for commercial applications. Using a window across the image to compute Fourier coefficients may be a way to gather more information to feed back into the classifier. The second recommendation is to fuse the results of two or more independent classifiers. Probabilities of samples being assigned to a certain class are weighted and a voting scheme could be used.

Appendix A. A LNKnet Helper

This appendix is a quick tutorial on the the program LNKnet. To start the program from your directory, put the following statement in the .cshrc file: set path=(path /home/cub7/LNKnet/bin). Now you should be set to call LNKnet. Do this by typing 'LNKnet' on the command line. A window will appear which is titled, "Experimental Control." If you have not obtained a copy of the help manual, do so, because it explains the various control parameters. The following tips will help.

- Call LNKnet from the directory that is one directory above where the data you want to use is located. For example, if you have the data files called xor.train and xor.test located in the directory called lnknet/data, then go to the lnknet directory on the command line and call up LNKnet.
- The top of the experimental control lists the parameter ALGORITHM. This allows you to choose a classification method.
- Directly below ALGORITHM is Algorithm Params... in which you can define the specific parameters of the classification technique.
- Under FILE NAMES: Just enter a name for Exper. Name and hit return and the other file names will be generated automatically. For Exper. Path enter 'data/' followed by a carriage return if the data is set up the same way as described in the first hint. All the files generated when the program runs will be put in the directory designated by the Exper. Path.
- For STANDARD DATA SET, enter NONE, unless you want to use the standard data sets found in the home/cub7/LNKnet directory.
- For Data Path, enter the directory the data is found in ie., 'data' for the example above. For data file prefix enter 'xor' for the example above. It will search for xor.train and xor.test. The number of features and output classes is obvious.
- Then go to the left side of the experimental control and enter the number of training patterns and test patterns. Click on the box next to the desired function whether it be train, test on training data, eval, or test.

- At this point hit START and you are on your way. The results will be outputted to the command tool and also to the log file.
- When entering parameters, be sure to hit a carriage return. The following is a script file which will run LNKnet's multilayer perceptron, testing after each training epoch. Depending on the particular problem you have, several parameters will need to be changed. The man pages for LNKnet list all the parameters.
- LNKnet needs a certain format for the data file. Separate each pattern by a carriage return and separate each feature by a space. To indicate the class of each pattern, place an integer before the first feature of each pattern. For example: 0 24.3 35.6 .....cr. The first class begins with a zero and so class 2 would begin with a 1 and so on.

That is a quick summary of the basics you will need to run the program. Most run-time errors occur because the program can't find the data files or the data files are in the wrong format.

When LNKnet is executed, it runs a series of script files which is puts in the log file. These script files can be edited to fit your desires. For example if you want to test after each epoch of training, generate a script file to do this. The following script file is an example of how this is done.

```
#!/bin/csh
ftrs/2c03mlp.run
set loc='pwd'
set epochs_left = 25
(time mlp \
-train  -create  -pathexp $loc  -ferror 2c03mlp.err.train  -fparam 2c03mlp.param\
 -fpid 2c03mlp.pid  -pathdata /tmp_mnt/home/hawkeye7/gshartle/lnknet/ftrs/\
 -finput test.train  -fdescribe test.train.defaults\
 -nraw 10  -npatterns 18  -normalize  -fnorm test.norm.simple\
 -cross_valid 0  -fcross_valid test.train.cv  -random  -seed 0\
 -priors_npatterns 18  -debug 0  -verbose 3  -verror 2  -nodes 10,50,2\
 -alpha 0  -etta 0.1  -epsilon 0.1  -kappa 0.01  -decay 0  -tolerance 0.2\
 -hfunction 0  -ofunction 0  -param 1 -criterion 0  -epochs 1  -batch 1,1,0\
  ) \
 |& nn_tee -h  2c03mlp.log
# test after first epoch
```

```
(time mlp -create \
-pathexp $loc  -ferror 2c03mlp.err.test  -fparam 2c03mlp.param\
 -fpid 2c03mlp.pid  -pathdata /tmp_mnt/home/hawkeye7/gshartle/lnknet/ftrs\
 -finput test.test  -fdescribe test.test.defaults  -nraw 10\
 -npatterns 18  -normalize  -fnorm test.norm.simple  -cross_valid 0\
 -fcross_valid test.test.cv  -random  -seed 0  -priors_npatterns 18\
 -debug 0  -verbose 3  -verror 2  -nodes 10,50,2  -alpha 0.7  -etta 0.05\
 -epsilon 0.1  -kappa 0.01  -decay 0  -tolerance 0.2  -hfunction 0\
 -ofunction 0  -param 1  -criterion 0  -epochs 25  -batch 1,1,0   ) \
 |& nn_tee -h -a 2c03mlp.log
@ epochs_left -= 1
while($epochs_left > 1)
echo $epochs_left
#train each remaining epoch
(time mlp \
-train  -pathexp $loc  -ferror 2c03mlp.err.train  -fparam 2c03mlp.param\
 -fpid 2c03mlp.pid  -pathdata /tmp_mnt/home/hawkeye7/gshartle/lnknet/ftrs/\
 -finput test.train  -fdescribe test.train.defaults\
 -nraw 10  -npatterns 18  -normalize  -fnorm a.norm.simple\
 -cross_valid 0  -fcross_valid test.train.cv  -random  -seed 0\
 -priors_npatterns 18  -debug 0  -verbose 3  -verror 2  -nodes 10,50,2\
 -alpha 0  -etta 0.1  -epsilon 0.1  -kappa 0.01  -decay 0  -tolerance 0.2\
 -hfunction 0  -ofunction 0  -param 1 -criterion 0  -epochs 1  -batch 1,1,0\
  ) \
 |& nn_tee -h -a 2c03mlp.log
# test after each epoch of training
(time mlp  \
-pathexp $loc  -ferror 2c03mlp.err.test  -fparam 2c03mlp.param\
 -fpid 2c03mlp.pid  -pathdata /tmp_mnt/home/hawkeye7/gshartle/lnknet/ftrs\
 -finput test.test  -fdescribe test.test.defaults  -nraw 10\
 -npatterns 18  -normalize  -fnorm test.norm.simple  -cross_valid 0\
 -fcross_valid test.test.cv  -random  -seed 0  -priors_npatterns 18\
 -debug 0  -verbose 3  -verror 2  -nodes 10,50,2  -alpha 0.7  -etta 0.05\
 -epsilon 0.1  -kappa 0.01  -decay 0  -tolerance 0.2  -hfunction 0\
 -ofunction 0  -param 1  -criterion 0  -epochs 25  -batch 1,1,0   ) \
 |& nn_tee -h -a 2c03mlp.log
@ epochs_left -=1
end
#train to get any remainder in the number of epochs
(time mlp \
-train  -pathexp $loc  -ferror 2c03mlp.err.train  -fparam 2c03mlp.param\
 -fpid 2c03mlp.pid  -pathdata /tmp_mnt/home/hawkeye7/gshartle/lnknet/ftrs/\
 -finput test.train  -fdescribe test.train.defaults\
```

```
 -nraw 10  -npatterns 18  -normalize  -fnorm test.norm.simple\
 -cross_valid 0  -fcross_valid test.train.cv  -random  -seed 0\
 -priors_npatterns 18  -debug 0  -verbose 3  -verror 2  -nodes 10,50,2\
 -alpha 0  -etta 0.1  -epsilon 0.1  -kappa 0.01  -decay 0  -tolerance 0.2\
 -hfunction 0  -ofunction 0  -param 1 -criterion 0  -epochs 1  -batch 1,1,0\
  ) \
 |& nn_tee -h -a 2c03mlp.log
# do final test
(time mlp  \
-pathexp $loc  -ferror 2c03mlp.err.test  -fparam 2c03mlp.param\
 -fpid 2c03mlp.pid  -pathdata /tmp_mnt/home/hawkeye7/gshartle/lnknet/ftrs\
 -finput test.test  -fdescribe test.test.defaults  -nraw 10\
 -npatterns 18  -normalize  -fnorm test.norm.simple  -cross_valid 0\
 -fcross_valid test.test.cv  -random  -seed 0  -priors_npatterns 18\
 -debug 0  -verbose 3  -verror 2  -nodes 10,50,2  -alpha 0.7  -etta 0.05\
 -epsilon 0.1  -kappa 0.01  -decay 0  -tolerance 0.2  -hfunction 0\
 -ofunction 0  -param 1  -criterion 0  -epochs 25  -batch 1,1,0   ) \
 |& nn_tee -h -a 2c03mlp.log
# plot results of Testing, change ferror to plot training
plot_perr -pathexp $loc -ferror 2c03mlp.err.test -fplot 2c03mlp.perr.plot \
 -autoscale -xmin 0 -xmax 10000 -ymin 0 -ymax 100 -xstep 1000 -ystep 10\
 -line_type 1  -trials 36\
 -title "Norm:Simple Net:10,50,2 Step:0.1 Momen:0.6"
echo "current directory:" >> 2c03mlp.log
echo $loc >> 2c03mlp.log
 plot rms error
plot_rmserr -pathexp $loc -ferror 2c03mlp.err.test -fplot 2c03mlp.rmserr.plot \
 -autoscale -xmin 0 -xmax 10000 -ymin 0 -ymax 100 -xstep 1000 -ystep 10\
 -line_type 1  -trials 75\
 -title "Norm:Simple Net:10,50,2 Step:0.1 Momen:0.6"
echo "current directory:" >> 2c03mlp.log
echo $loc >> 2c03mlp.log
```

Appendix B. Sourcecode

B.1 Scriptfiles

```
# Program to find selected cities in the cd rom
#!/bin/csh -f
setenv HIPSDIR /cdrom/train/cities/bd
set outdir = $HOME/cedar/data/train
set C = 0
set W = 0
set V = 0
unalias ls
cd $HIPSDIR
touch ~/temp/hips.junk
foreach j (bd*)
cd $j
echo $j
set l = `ls -a bd*.*`
foreach i ( $l )
echo $i
~/cedar/bin/deltau < $HIPSDIR/$j/$i | ~/cedar/bin/hdinfo | grep Truth > ~/temp/junk.hips
set a = `ls ~/temp/junk.hips`

set yn = `grep 'Image' $a | awk '/[Vv][Ee][Gg][Aa][Ss]/ {print "y"}'`
if ( $yn == "y" ) then
~/cedar/bin/deltau < $i | ~/cedar/bin/hips2sun > $outdir/1/$i;
 set V = (`echo "$V 1 + p" | dc`)
echo 2,""  >>$outdir/temp/classes
set k=`grep Image $a`
echo $k"    " $i"    " "tally = "$V, "    class = 1", >>$outdir/temp/tally
else
endif
set yn = "n"



set yn = `grep 'Image' $a | awk '/[Ww][Aa][Ss][Hh][Ii][Nn][Gg][Tt][Oo][Nn]/ {print "y" }'`
if ( $yn == "y" ) then
~/cedar/bin/deltau < $i | ~/cedar/bin/hips2sun > $outdir/2/$i;
set W = (`echo "$W 1 + p" | dc`)
echo 3,"" >>$outdir/temp/classes
set k=`grep Image $a`
echo $k"    "$i"    " "tally = "$W, "     class = 2">>$outdir/temp/tally
```

```
else
endif
set yn = "n"


set yn = `grep 'Image' $a | awk '/[cC][Ii][Tt][Yy]/ {print "y"}'`
if ( $yn == "y" ) then
~/cedar/bin/deltau < $i | ~/cedar/bin/hips2sun > $outdir/3/$i;
set C = (`echo "$C 1 + p" | dc`)
echo 4,"" >>$outdir/temp/classes
set k=`grep Image $a`
echo $k"    "$i"    " "tally = "$C, "     class = 3">>$outdir/temp/tally
else
endif
set yn = "n"



end
cd ..
end

# Program to preprocess the image

#!/bin/csh -f
#foreach i (b*.[0-9])


# Window the binarized image and do the dft

# convert the raster file of the image to viff format,
# do a histogram strech and threshold.
tiff2viff -i $i -o /usr/tmp/two -v 0
#rast2viff -i $i -o /usr/tmp/one -p 0
#putimage -i /usr/tmp/one -update 2
#vhstr -i /usr/tmp/one -o /usr/tmp/two -p 0
vthresh -i /usr/tmp/two -o /usr/tmp/three -l 235 -v 255
#putimage -i /usr/tmp/three -update 2


viff2rast -i /usr/tmp/three -o test -p 0
rast2viff -i test -o /usr/tmp/rast2HAAa14313 -p 0
viff2pbm -i /usr/tmp/rast2HAAa14313 -o $i.asc -r 0
```

```
# convert the binarized image back to asc,
# crop the image, and do a dft

#viff2pbm -i /usr/tmp/three -o $i.asc -r 0
#~/bin/crop $i.asc $i.crop aux
~/bin/dft_all $i.asc bin_win_ftrs/$i.allpbw
#~/bin/dft_all $i.crop bin_win_ftrs/$i.allpbw
rm $i.asc
#rm $i.crop

# get the cropped dimensions from the .aux file

#set dims = `head -2 aux`

#echo $dims[1],$dims[2], $dims[3], $dims[4]
# Window the original image and do the dft to get the features

# crop the original image

#putimage -i /usr/tmp/one -update 2
#vextract -i /usr/tmp/one -o /usr/tmp/cut -x $dims[2] -y $dims[1] -w $dims[3] -h $dims[4]
#putimage -i /usr/tmp/cut -update 2

# convert from viff2pbm, run the dft routine to get the features from the unbinarized images

#viff2pbm -i /usr/tmp/cut -o $i.asc -r 0
#~/bin/dft_all $i.asc win_ftrs/$i.allpw
#rm $i.asc

end



# Program to do addon testing

#!/bin/csh -f



echo "" > result_mlp_49
echo "" > result_mlp_49_avgtest
echo "" > result_mlp_49_avgtrain
echo "" > result_mlp_49_avgrmstest
```

```
echo "" > result_mlp_49_avgrmstrain
echo "" > feature
echo features" "trials" "%error_test" "%error_train" "rms_err_test" "rms_err_train" "misses >> result_mlp_49

echo "" >> result_mlp_49

set bestfeature = `echo '0 0 0 0 0 '`


foreach j (1 2 3 4 5)

if ($j != 1) then
set best = `awk '{print $1}' feature`
echo $best
set k = `echo "$j 1 - p" | dc`
while($k > 0)
set bestfeature[$k] = $best[$k]
@ k -=1
end
endif
echo $bestfeature

echo " " > test_features



foreach i (221 222 220 199 243 201 241)

echo $i,$j

if ($bestfeature[1] != $i && $bestfeature[2] != $i
&& $bestfeature[3] != $i && $bestfeature[4] != $i
&& $bestfeature[5] != $i) then



echo "" >> result_mlp_49

set avgtest = 0
set avgtrain = 0
set avgrmstest = 0
set avgrmstrain = 0
```

```
set trial = 1



get_each_feature 4csamps.normid fomftrset 400 441 $i

rand4c fomftrset test.train test.test 400 $j list

~/lnknet/ftrs/whoosh3.run $j


# set up a file called error to write all the results of testing to.

set x = `sed s/"( "/"("/ 2c03mlp.log| grep Overall | awk '{print$4,$6}'`
#set x1 = `sed s/"( "/"("/ 2c03mlp.log| grep Overall | awk '{print$4,$6}'`

echo " " > error
echo $#x >> error
echo $x >> error

# get the lowest error

~/bin/geterror



set f = ` awk '{print $j}' feature`
set x1 = ` awk '{print $0}' error_report`
set y1 = ` awk '{print $0}' misclasslist`
set z1 = ` echo $#y1`

echo $f >> result_mlp_49
echo $i"  "$trial" "$x1[1]" "$x1[2]" "$x1[3]" "$x1[4]" " >> result_mlp_49

#echo $i"  "$trial" "$x1[1]" "$x1[2]" " >> result_mlp_49

while($z1 > 0)
echo " "$y1[$z1] >> result_mlp_49
@ z1 -=1
end
```

```
rand4c fomftrset test.train test.test 400 $j list

~/lnknet/ftrs/whoosh3.run $j


set x='sed s/"( "/"("/ 2c03mlp.log| grep Overall | awk '{print$4,$6}''

$set x2 = 'sed s/"( "/"("/ 2c03mlp.log| grep Overall | awk '{print$4,$6}''

echo " " > error
echo $#x >> error
echo $x >> error
set trial = 2
~/bin/geterror

set x2 = ' awk '{print $0}' error_report'
set y2 = ' awk '{print $0}' misclasslist'
set z2 = ' echo $#y2'

echo $f >> result_mlp_49
echo $i"  "$trial" "$x2[1]" "$x2[2]" "$x2[3]" "$x2[4]" " >> result_mlp_49

#echo $i"  "$trial" "$x2[1]" "$x2[2] >> result_mlp_49

while($z2 > 0)
echo " "$y2[$z2] >> result_mlp_49
@ z2 -=1
end



rand4c fomftrset test.train test.test 400 $j list

~/lnknet/ftrs/whoosh3.run $j


set x='sed s/"( "/"("/ 2c03mlp.log| grep Overall | awk '{print$4,$6}''
set x3='sed s/"( "/"("/ 2c03mlp.log| grep Overall | awk '{print$4,$6}''

echo " " > error
echo $#x >> error
```

```
echo $x >> error
set trial = 3
~/bin/geterror

set x3 = ` awk '{print $0}' error_report`
set y3 = ` awk '{print $0}' misclasslist`
set z3 = ` echo $#y3`

echo $f >> result_mlp_49
echo $i"  "$trial" "$x3[1]" "$x3[2]" "$x3[3]" "$x3[4]" " >> result_mlp_49

#echo $i"  "$trial" "$x3[1]" "$x3[2]" " >> result_mlp_49

while($z3 > 0)
echo " "$y3[$z3] >> result_mlp_49
@ z3 -=1
end

set avgtest = `echo "2 k" $x1[1] $x2[1] $x3[1]"++ 3 / p"|dc`
set avgtrain = `echo "2 k" $x1[2] $x2[2] $x3[2]"++ 3 / p"|dc`
set avgrmstest = `echo "2 k" $x1[3] $x2[3] $x3[3]"++ 3 / p"|dc`
set avgrmstrain = `echo "2 k" $x1[4] $x2[4] $x3[4]"++ 3 / p"|dc`

echo "" >> result_mlp_49
echo "" >> result_mlp_49
echo " "avg" "$avgtest" "$avgtrain" "$avgrmstest" "$avgrmstrain >> result_mlp_49
echo "" >> result_mlp_49
echo "" >> result_mlp_49

echo $i"  "$avgtest >> result_mlp_49_avgtest
echo $i"  "$avgtrain >> result_mlp_49_avgtrain
echo $i"  "$avgrmstest >> result_mlp_49_avgrmstest
echo $i"  "$avgrmstrain >> result_mlp_49_avgrmstrain

echo $i"  "$avgtest >> test_features


rm 2c03mlp.log
rm 2c03mlp.err.test
endif

end
```

```
~/bin/sort_best_feature



end

# Program to find the misclassified samples

#!/bin/csh -f

# read in the misclassified samples

set number = 'awk '{print $1}' miss'
set desired = 'awk '{print $3}' miss'
set classified = 'awk '{print $5}' miss'

# write them to a file

echo > miss1
echo $#number >> miss1
echo $number >> miss1
echo $desired >> miss1
echo $classified >> miss1


# it calls the program miss_patterns to get the missed patterns
# and find the filename of the pattern so it can display it.

miss_patterns

#set patterns = 'awk '{print $1}' misclasslist'

#foreach i ($patterns)
#rast2viff -i ../../cedar/data/train/all_patterns/$i -o /usr/tmp/$i -p 0
#putimage -i /usr/tmp/$i -update 2
#end
```

B.2 C code

```
/*************************************************
     Compute the 2dft of
    a NxM array of values

   This program reads an ascii file
   that has been generated by
   converting a raster file to pbm in
   khorus.

   It reads every third value in the
   array to account for the fact that
   the conversion was greyscale.

*************************************************/

#include <stdio.h>
#include <math.h>

main (argc, argv)
int argc;
char *argv[];
{

/****   true_height is the real array dim in y space
       true_width is the real array dim in x space  *******/

     int true_height, true_width, ORDER, FILTER,x,y,i,j,k,l,waste2;
      float  high, across, tempk, templ, cossin_term, norm,dc_term;
      float coeff[21][21];
   int city_name[1000][1000];
      int data[1000][1000], junk;
      char waste;

      FILE *in_file, *out_file;
         if ((in_file=fopen(argv[1], "r"))==NULL)
         printf("can't open\n");


   out_file = fopen(argv[2],"w");


/** read header info ***/

fscanf (in_file,"%s", &waste);
fscanf (in_file,"%d", &true_width);
printf(" %d ",true_width);
fscanf (in_file,"%d", &true_height);
printf(" %d  ",true_height);
    if (true_width≥1000)
       printf("out of bounds\n");
```

```
    if (true_height≥1000)
       printf("out of bounds");
fscanf (in_file,"%d", &waste2);

/**  read in the matrix of values, every third value  ***/

 for (x=0; x<true_height; x++)
{

      for (y=0; y<true_width; y++)
       {
         fscanf (in_file,"%d %d %d", &data[x][y], &junk, &junk);

         city_name[x][y]=data[x][y];
       }
}


high = 2*M_PI/(true_height);
across = 2*M_PI/(true_width);


/****  FILTER = ORDER*2+1   where order is the number of harmonics ***/


ORDER = 10;
FILTER = ORDER*2+1;

for (k=0; k<FILTER; k++)
for (l=0; l<FILTER; l++)

    coeff[k][l] = 0.0;



for (k=0; k<ORDER; k++)

   for (l=0; l<FILTER; l++)
       {
   tempk = (k−ORDER)*high;
   templ = (l−ORDER)*across;


   for (i=0; i<true_height; i++)

   for (j=0; j<(true_width); j++)
    {
     cossin_term = −i*tempk−j*templ;
     coeff[k][l] += city_name[i][j] * cos(cossin_term);
     coeff[20−k][l] += city_name[i][j] * sin(cossin_term);

    }
```

```
  }

k = ORDER;
for (l=0; l<ORDER; l++)
 {
  templ = (l-ORDER)*across;

   for (i=0; i<true_height; i++)
   for (j=0; j<true_width; j++)
     {
       cossin_term = -j*templ;
       coeff[k][l] += city_name[i][j] * cos(cossin_term);
       coeff[k][20-l] += city_name[i][j] * sin(cossin_term);

     }
 }

dc_term = 0;
for (i=0; i<true_height; i++)
for (j=0; j<true_width; j++)

      dc_term += city_name[i][j];
printf("%f",dc_term);
coeff[ORDER][ORDER]=dc_term;


/************** ENERGY NORMALIZE ***********************/

     norm = 0.0;
     for (k=0; k<FILTER; k++)
       for (l=0; l<FILTER; l++)
       norm += coeff[k][l]*coeff[k][l];

   norm = sqrt(norm);


   for (k=0; k<FILTER; k++)
   for (l=0; l<FILTER; l++)
      coeff[k][l] = coeff[k][l]/norm;

/************** WRITE COEFFICIENTS TO FILE *************/
      for (k=0; k<FILTER; k++)
      for (l=0; l<FILTER; l++)

   fprintf(out_file,"%f\n",coeff[k][l]);


      fclose(in_file);
      fclose(out_file);

} /***  END MAIN  ***/
```

```
/************************************************/

 Program: crop.c

 Description: Crops a binarized image

************************************************/
#include <stdio.h>
#include <math.h>

main (argc, argv)
int argc;
char *argv[];
{


/****   true_height is the real array dim in y space  ***
       true_width is the real array dim in x space  *******/

     int true_height, true_width, ORDER, FILTER,x,y,i,j,k,l,waste2;
   int city_name[1000][1000];
      int data[1000][1000], junk;
      int top, bottom, left_side, right_side,width, height;
      int count=0, val;
     char waste[5];

     FILE *in_file, *out_file, *out_file2;
        if ((in_file=fopen(argv[1], "r"))==NULL)
        printf("can't open\n");


  out_file = fopen(argv[2],"w");
     out_file2 = fopen(argv[3],"w");


fscanf (in_file,"%s", waste);
fscanf (in_file,"%d", &true_width);
printf(" %d ",true_width);
fscanf (in_file,"%d", &true_height);
printf(" %d   ",true_height);
   if (true_width≥1000)
      printf("out of bounds\n");
   if (true_height≥1000)
      printf("out of bounds");
fscanf (in_file,"%d", &waste2);

/**  read in the matrix of values, every third value  ***/
top = -1;
 for (x=0; x<true_height; x++)
  {
```

```
        for (y=0; y<true_width; y++)
         {
           fscanf (in_file,"%d %d %d", &data[x][y], &junk, &junk);

           city_name[x][y]=data[x][y];
        if ((top == -1) && (city_name[x][y] == 0) && (y ≠ 0))
              top = x;
         }
    }

    left_side = -1;
    for (y=1; (y < true_width) && (left_side==-1); y++)
      for (x=top; x < true_height; x++)
        if (city_name[x][y] == 0)
          {left_side = y; break;}
    bottom = -1;
     for (x=true_height-1; (x > top) && (bottom == -1); x--)
       for (y=left_side; y < true_width ; y++)
         if (city_name[x][y] == 0)
           {bottom=x; break;}

    right_side = -1;
     for (y=true_width-1; (y > left_side) && (right_side == -1); y--)
       for (x=top; x < bottom; x++)
      if (city_name[x][y] == 0)
            {right_side=y; break;}

width = right_side - left_side + 1;
height = bottom - top + 1;
fprintf(out_file,"%s\n%d %d\n%d\n",waste, width, height,waste2);
fprintf(out_file2,"%d %d %d %d",top,left_side,width,height);
for (x=top;x≤bottom;x++)
  for (y=left_side; y≤right_side;y++)
   {
    val = city_name[x][y];
    fprintf(out_file,"%3d %3d %3d   ", val, val, val):
    if (++count == 4)
    {
       fprintf("\n");
       count = 0;
    }
   }

fclose(in_file);
fclose(out_file);

}
#include <stdio.h>
#define SQR(x) (x)*(x)

/*
```

```
 * normalize: normalize input features based on nsamples or (nsamples-1)
 *          samples
 *
 * Inputs:
 *   data: nsamples x dim array of data--each row is a sample
 *   nsamples: number of training samples
 *   dim: number of elements in each training vector (before augmenting)
 *
 * Output (returned):
 *   normal: copy of data with all features in all samples normalized
 */

main(argc, argv)
int argc;
char *argv[];
{

/*  Matrix normalize(Matrix data, int nsamples, int dim)  */
{
   int    i, j,num_patterns, num_features;
   float sum, sumsq, data[1200][1200], mean[1200], std[1200];
   FILE *infile, *outfile;

/*   Vector mean, std;  */

if (argc != 5)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile> <number of classes> <vectors per class>
< number of features > < number of patterns>\n", argv[0], argv[0]);
      exit(1);
   }

   if ((infile = fopen(argv[1], "r")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n", argv[1]);
   exit(0);
   }
   if ((outfile = fopen(argv[2], "w")) == NULL)
   {
   printf(stderr, "Couldn't open output file %s\n", argv[2]);
   exit(0);
}

num_features = atoi(argv[3]);
num_patterns = atoi(argv[4]);

for(i=0; i < num_patterns; i++)
 for(j=0; j< num_features; j++)
  fscanf(infile,"%f",&data[i][j]);
```

```
/*    mean = v_alloc(dim);
      std = v_alloc(dim);  */

      /* Compute mean and std vector: */
      for (j=0; j < num_features; j++)
      {
         sum = 0.0;
         sumsq = 0.0;
         for (i=0; i < num_patterns; i++)
         {
            sum += data[i][j];
            sumsq += SQR(data[i][j]);
      }

         mean[j] = sum / num_patterns;
         std[j] = (sumsq - (SQR(sum) / num_patterns))/(num_patterns-1);
      }

      /*  Normalize and augment all   ta samples:  */
      for (i=0; i < num_patterns; i++)
      {
         if (i ≠ 0) fprintf(outfile,"\n");

      for (j=0; j < num_features; j++)
            {
         data[i][j] = (data[i][j] - mean[j]) / std[j];
            fprintf(outfile,"%f ",data[i][j]);
         }

      }

/*    free(mean);
      free(std);  */

}

}
/************************************************

   Program: separate.c


    Description:  Assigns a figure of merit
    for each dimension in
    n-dimensional feature space



************************************************/

#include <stdio.h>
```

```
#include <math.h>
#include "jkmacros.h"

main(argc, argv)
int argc;
char *argv[];
{
int i,j,k;
int num_classes, length, vectors_per_class, num_features, num_patterns;
int feature_number[500];
float class_mean[500][500],class_variance[500][500],temp1,temp2;
float newdata_matrix[500][500],across_class_mean[500];
float across_class_variance[500], mean_of_var[500];
float var_of_means[500], fom_ordered_vector[500],fom[500];
FILE *infile, *outfile;

 if (argc ≠ 7)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile> <number of classes> <vectors per class>
< number of features > < number of patterns>\n", argv[0], argv[0]);
      exit(1);
   }

   if ((infile = fopen(argv[1], "r")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n", argv[1]);
   exit(0);
   }
   if ((outfile = fopen(argv[2], "w")) == NULL)
   {
   printf(stderr, "Couldn't open output file %s\n", argv[2]);
   exit(0);
}

num_classes = atoi(argv[3]);
vectors_per_class = atoi(argv[4]);
num_features = atoi(argv[5]);
num_patterns = atoi(argv[6]);

/* read in the data file so that the first column is class 1, feature
  vector 1; the second column is feature vector 2 and so on. */

for(i=1; i ≤ num_patterns; i++)
 for(j=1; j≤ num_features; j++)
   fscanf(infile,"%f",&newdata_matrix[j][i]);


/*************************************************************************
Calculate class_mean and class_variance matrices
*************************************************************************/
```

```
Loop1i(num_classes)
   Loop1j(num_features)
      {
      temp1 = 0.0;
      Loop1k(vectors_per_class)
         {
         temp1 += newdata_matrix[j][k + ((i - 1) * vectors_per_class)];
         }
      class_mean[j][i] = temp1/vectors_per_class:
/* printf("class_mean %f\n",class_mean[j][i]);
fprintf(outfile,"%f\n",class_mean[j][i]);  */
      }

Loop1i(num_classes)
   Loop1j(num_features)
      {
      temp2 = 0.0:
      Loop1k(vectors_per_class)
         {
            temp2 += (newdata_matrix[j][k + ((i - 1) * vectors_per_class)] - class_mean[j][i]) * (new-
data_matrix[j][k + ((i - 1) * vectors_per_class)] - class_mean[j][i]) ;
         }
      class_variance[j][i] = temp2/vectors_per_class;
/*printf("class var %f\n", class_variance[j][i]);
fprintf(outfile,"%f\n",class_variance[j][i]); */
      }


/*************************************************************************
Calculate across_class_mean and across_class_variance matrices
*************************************************************************/

Loop1i(num_features)
   {
   temp1 = 0.0;
   Loop1j(num_classes)
      temp1 += class_mean[i][j];
   across_class_mean[i] = temp1/mum_classes;
   }

Loop1i(num_features)
   {
   temp2 = 0.0;
   Loop1j(num_classes)
      temp2 += (class_mean[i][j] - across_class_mean[i]) * (class_mean[i][j] - across_class_mean[i]);
   across_class_variance[i] = temp2/mum_classes;
   }

/*************************************************************************
Calculate mean_of_var and var_of_mean vectors
*************************************************************************/
```

```
Loop1i(num_features)
   {
   temp1 = 0.0;
   temp2 = 0.0;
   Loop1j(num_classes)
      {
      temp1 += class_variance[i][j];
      temp2 += (class_mean[i][j] - across_class_mean[i]) * (class_mean[i][j] - across_class_mean[i]);
      }
   mean_of_var[i] = temp1/num_classes;
   var_of_means[i] = temp2/num_classes;
   }

/*****************************************************************************
Calculate (Figure of Merit) fom vector
*****************************************************************************/


Loop1i(num_features)
   {fom_ordered_vector[i] = fom[i] = var_of_means[i]/mean_of_var[i];
      feature_number[i]=i;

   }

/*****************************************************************************
Sort
*****************************************************************************/
piksr2(num_features,fom_ordered_vector, feature_number);

Loop1i(num_features)
fprintf(outfile."%d\t%f\n",feature_number[(num_features+1)-i],fom_ordered_vector[(num_features+1)-i]);

}/*end main*/

/*****************************************************************************

  Program: klt.c

  Description: Program to calculate the eigenvectors of
       a set of n-dimensional data



*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "jkmacros.h"

main(argc, argv)
int argc;
```

```
char *argv[];
{

FILE    *infile, *outfile, *outfile2;
int length,num_train,num_codewords,num_classes, num_eigvectors;
int i,j,N,k,M,nrot;
float     **matrix(),  *vector(), **A, **A_trans, **u, **L, **v, *d, *average_temp, temp;
void free_vector(), free_matrix(),eigsrt(),jacobi();
char type_name[30],avg_file[30], msg[30], msg1[30],filename[40],file[40];



length=atoi(argv[3]);
num_train=atoi(argv[4]);
num_eigvectors = atoi(argv[5]);

 if (argc ≠ 7)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
      exit(1);
   }

   if ((infile = fopen(argv[1], "r")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n", argv[1]);
   exit(0);
   }
   if ((outfile = fopen(argv[2], "w")) == NULL)
   {
   printf(stderr, "Couldn't open output file %s\n", argv[2]);
   exit(0);
}
   if ((outfile2 = fopen(argv[6], "w")) == NULL)
   {
   printf(stderr, "Couldn't open output file %s\n", argv[6]);
   exit(0);
}


/****** Allocate memory ******/

A = matrix(1,length,1,num_train);
A_trans = matrix(1,num_train,1,length);
average_temp = vector(1,length);
L = matrix(1,num_train,1,num_train);
d = vector(1,num_train);
v = matrix(1,num_train,1,num_train);

/****** Initalize matrix and vectors ******/

for(j=1;j≤num_train;j++)
   for(i=1;i≤length;i++)
```

```
      A_trans[j][i]=A[i][j]=average_temp[i]=0.0;



   printf("\nThe users being trained on are :\n\n");
/*   open_read(argv[1], "2c03.train");  */
   Loop1i(num_train)
      {
           Loop1j(length)
         {
         fscanf(infile, "%f", &A[j][i]);
         }
      }
/***************Calculate average vector*******************/

/* sprintf(avg_file, "avg_%s.dat", type_name); */
/* open_write(arg[4], avg_file);  */
Loop1i(length)
   {
   temp = 0.0;
   Loop1j(num_train)
      {
      temp += A[i][j];
      }
   average_temp[i] = temp/mum_train:

   }


/**************Subtract average vector************************/

Loop1j(num_train)
   Loop1i(length)
      A[i][j] = A[i][j] - average_temp[i];

free_vector(average_temp, 1, length);

/*************Make transpose matrix************************/

Loop1j(num_train)
   Loop1i(length)
      A_trans[j][i] = A[i][j];



/******************Matrix multiply A by itself***************/

Loop1i(num_train)
   Loop1j(num_train)
      {
      temp = 0.0;
```

```
      Loop1k(length)
         temp = temp + A_trans[i][k] * A[k][j];
      L[i][j] = temp;
      }
free_matrix(A_trans, 1, num_train, 1, length);

/***************Do Jacobi rotation and sort eigenstuff***********/

jacobi(L, num_train, d, v, &nrot);
eigsrt(d, v, num_train);
for (i=1; i<num_eigvectors; i++)
  { printf("eigenvalue  %d  is %f\n",i,d[i]);
   fprintf(outfile2,"%f\n",d[i]); }
/*****************Find eigenvectors*********************************/

u = matrix(1, length, 1, num_train);
Loop1i(num_train)
   Loop1j(length)
      u[j][i] = 0.0;

Loop1i(num_train)
   Loop1j(num_train)
      Loop1k(length)
         u[k][i] = v[j][i] * A[k][j] + u[k][i];


/***********Write file containing list of eigenvector names*********/

 /* sprintf(msg, "%s_train.out", type_name); */
 /* open_write(outfile, msg);  */
 /* sprintf(msg1, "eigenvector", type_name); */
/*  Loop1i(num_eigvectors)
    {
    sprintf(file, "%s%d.dat", msg1, i);
    fprintf(outfile, "%s\n", file);
    }
 fclose(outfile);  */

/* close this for selected eigenvectors */

Loop1i(num_eigvectors)
   {

   Loop1j(length)
      fprintf(outfile, "%g\n", u[j][i]);

   }
fclose(outfile);

/* to print selected eigenfeatures

for(i=1; i ≤ num_eigvectors; i++)
```

```
   for(j=1; j ≤ length; j++)
    { if (i==1) fprintf(outfile,"%g\n", u[j][i]);
   if (i==5) fprintf(outfile,"%g\n", u[j][i]);
    if (i==9) fprintf(outfile,"%g\n", u[j][i]);
      if (i==12) fprintf(outfile,"%g\n", u[j][i]);
   if (i==17) fprintf(outfile,"%g\n", u[j][i]);
     if (i==7) fprintf(outfile,"%g\n", u[j][i]);
   if (i==3) fprintf(outfile,"%g\n", u[j][i]);
   if (i==4) fprintf(outfile,"%g\n", u[j][i]);
     if (i==3) fprintf(outfile,"%g\n", u[j][i]);
   if (i==24) fprintf(outfile,"%g\n", u[j][i]);      } */


free_matrix(A, 1, length, 1, num_train);
free_matrix(u, 1, length, 1, num_train);
free_matrix(A, 1,length,1,num_train);
free_matrix(L, 1,num_train,1,num_train);
free_matrix(v,1,num_train,1,num_train);
free_vector(d, 1,num_train);


} /* end main*/

/************************************************

  Program: kltftrs.c

  Description:  Program which takes the
  the original data set and multiplies the
  eigenvectors calculated from klt.c to get
  the new set of features

************************************************/
#include <stdio.h>

main(argc, argv)
int argc;
char *argv[];
{

int i,j,k, num_eigenvec, num_features,num_patterns;
float input[500][500], kltvec[500][500], sum[500][500], feat;

FILE    *infile, *infile2, *outfile;


 if (argc ≠ 7)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
      exit(1);
   }
```

```
    if ((infile = fopen(argv[1], "r")) == NULL)
    {
    printf(stderr, "Couldn't open file list %s\n", argv[1]);
    exit(0);
    }

    if ((infile2 = fopen(argv[2], "r")) == NULL)
    {
    printf(stderr, "Couldn't open file list %s\n", argv[2]);
    exit(0);
    }
    if ((outfile = fopen(argv[3], "w")) == NULL)
    {
    printf(stderr, "Couldn't open output file %s\n", argv[3]);
    exit(0);
}


num_eigenvec = atoi(argv[4]);
num_features = atoi(argv[5]);
num_patterns = atoi(argv[6]);


/** read in the values of the klt matrix **/
for (i=0; i < num_eigenvec; i++)
  for (j=0; j < num_features; j++)
    fscanf (infile2, "%f", &kltvec[i][j]);

/** read in the data set to be reduced **/
for (i=0; i < num_patterns; i++)
   for (j=0; j < num_features; j++)
     fscanf (infile, "%f", &input[i][j]);

/** establish the new set of features vector **/
for (i=0; i < num_patterns; i++)
  for(j=0; j < num_eigenvec; j++)
     sum[i][j]=0.0;


/** multiply input and klt to get new features **/
for (i=0; i < num_patterns; i++)
 for (k = 0; k < num_eigenvec; k++)
   for (j=0; j < num_features; j++)
{
      feat = input[i][j] * kltvec[k][j];
      sum[i][k] = sum[i][k] + feat;
}

/***   write to a file  ***/
for (i=0; i < num_patterns; i++)
 {      if (i ≠ 0) fprintf(outfile, "\n");
  for (j=0; j < num_eigenvec; j++)
```

```
    fprintf(outfile, "%f ",sum[i][j]);
  }
  fclose(infile);
  fclose(infile2);
  fclose(outfile);

  } /* end main */

  /************************************************/

   Program: magphase.c

   Description: Computes magnitude and phase
   of the lower 3 harmonics

  /************************************************/



  #include <stdio.h>
  #include <math.h>
  #define ARC(a) (float)atan((double)(a))
  #define PI 3.141592654
  main (argc, argv)
  int argc:
  char *argv[]:
  {
     int i,j, k, junk, num_patterns, num_features;
     float mag[500][10][10], phase[500][10][10], data[500][10][10];
     float temp,temp1,temp2;
     FILE *file, *phase_file, *mag_file, *phase_only_file, *mag_phase_file, *mag_only_file;


     if (argc ≠ 2)
     {
        fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
        exit(1);
     }

     if ((file = fopen(argv[1], "r")) == NULL)
     {
     printf(stderr, "Couldn't open file list %s\n", argv[1]);
     exit(0);
     }

  num_patterns = 400;
  num_features = 49;
```

```
for (k=1; k ≤ num_patterns; k++)
 { fscanf(file,"%d", &junk);

 for (i=1; i ≤7; i++)
  for (j=1; j ≤ 7; j++)
        fscanf(file, "%f",&data[k][i][j]); }


for (k=1; k≤ num_patterns; k++)

for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
mag[k][i][j] = 0.0;
phase[k][i][j] = 0.0;


for (k=1: k≤ num_patterns: k++)
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)

    mag[k][i][j] = sqrt(data[k][i][j]*data[k][i][j] + data[k][8−i][j]*data[k][8−i][j]);


i=4;
for (j=1; j≤3; j++)

mag[k][i][j] = sqrt(data[k][i][j]*data[k][i][j] + data[k][i][8−j]*data[k][i][8−j]);

}




for (k=1; k≤ num_patterns; k++)
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)


if (data[k][i][j] > 0 && data[k][8−i][j] ≥0)
  phase[k][i][j] = (180/PI)*ARC((double)data[k][8−i][j]/data[k][i][j]);

else if (data[k][i][j] > 0  && data[k][8−i][j] < 0)
  phase[k][i][j] = 360 + (180/PI)*ARC((double)data[k][8−i][j]/data[k][i][j]);

else if (data[k][i][j] < 0 && data[k][8−i][j] ≥0)
  phase[k][i][j] = 180 + (180/PI)*ARC((double)data[k][8−i][j]/data[k][i][j]);

else if (data[k][i][j] < 0 && data[k][8−i][j] < 0)
  phase[k][i][j] = 180 + (180/PI)*ARC((double)data[k][8−i][j]/data[k][i][j]);
```

```
i=4;
for (j=1; j≤3; j++)

if (data[k][i][j] > 0 && data[k][i][8−j] ≥0)
  phase[k][i][j] = (180/PI)*ARC((double)data[k][i][8−j]/data[k][i][j]);

else if (data[k][i][j] > 0  && data[k][i][8−j] < 0)
  phase[k][i][j] = 360 + (180/PI)*ARC((double)data[k][i][8−j]/data[k][i][j]);

else if (data[k][i][j] < 0 && data[k][i][8−j] ≥0)
  phase[k][i][j] = 180 + (180/PI)*ARC((double)data[k][i][8−j]/data[k][i][j]);

else if (data[k][i][j] < 0 && data[k][i][8−j] < 0)
  phase[k][i][j] = 180 + (180/PI)*ARC((double)data[k][i][8−j]/data[k][i][j]);

}


/* print to a file to verify it works */

phase_file = fopen("phase","w");
k=1;
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
fprintf(phase_file, "%f\n", phase[k][i][j]);

i=4;
for (j=1; j ≤3; j++)
fprintf(phase_file, "%f\n", phase[k][i][j]);

}

mag_file = fopen("magnitude","w");
k=1;
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
fprintf(mag_file, "%f\n", mag[k][i][j]);

i=4;
for (j=1; j ≤3; j++)
fprintf(mag_file, "%f\n", mag[k][i][j]);
}


/* print a new feature set */

/* print mag and phase */

mag_phase_file = fopen("mag_phase_ftrs","w");
```

```
for (k=1; k≤ num_patterns; k++)
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
fprintf(mag_phase_file, "%f ", mag[k][i][j]);

i=4;
for (j=1; j ≤3; j++)
fprintf(mag_phase_file, "%f ", mag[k][i][j]);

j=4;   /*print dc term */
fprintf(mag_phase_file, "%f ", data[k][i][j]);

for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
fprintf(mag_phase_file, "%f ", phase[k][i][j]);

i=4;
for (j=1; j ≤3; j++)
fprintf(mag_phase_file, "%f ", phase[k][i][j]);
fprintf(mag_phase_file, "\n");
}


/* print phase only */

phase_only_file = fopen("phase_ftrs","w");

for (k=1; k≤ num_patterns; k++)
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
fprintf(phase_only_file, "%f ", phase[k][i][j]);

i=4;
for (j=1; j ≤3; j++)
fprintf(phase_only_file, "%f ", phase[k][i][j]);
fprintf(phase_only_file, "\n");

}

/* print mag only */

mag_only_file = fopen("mag_ftrs","w");

for (k=1; k≤ num_patterns; k++)
{
for (i=1; i≤3; i++)
  for (j=1; j ≤ 7; j++)
fprintf(mag_only_file, "%f ", mag[k][i][j]);
```

```
i=4;
for (j=1; j ≤3; j++)
fprintf(mag_only_file, "%f ", mag[k][i][j]);

j=4;   /*print dc term */
fprintf(mag_only_file, "%f ", data[k][i][j]);



}




} /** end main **/
/*********************************************/

  Program: getfom.c

  Description:  Builds a feature set from
  selected features of the 441 calculated
  from the original feature set.

*********************************************/

#include<stdio.h>

main(argc, argv)
int argc;
char *argv[];
{
    int i, j, done = 0, num_patterns, num_features,junk,dimensions;
    float val,data[800][800];
    char filename[80], temp[20];
    FILE *file, *outfile;

     static int class[] =

    {0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

```
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2. 2. 2. 2, 2,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2. 2, 2, 2, 2. 2. 2, 2,
      3, 3, 3, 3, 3. 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
      3, 3. 3, 3, 3, 3, 3, 3. 3, 3, 3. 3, 3. 3, 3, 3. 3. 3. 3, 3,
      3, 3. 3, 3, 3, 3, 3, 3. 3, 3, 3, 3, 3. 3, 3, 3. 3. 3, 3, 3,
      3. 3. 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3. 3, 3, 3,
      3. 3, 3, 3, 3, 3, 3, 3. 3, 3, 3, 3, 3. 3, 3, 3, 3. 3, 3, 3  };

/* added an extra zero in class[] to account for the for loop below
  beginning with i=1   */

 static int selected_features [] =
{ 221, 222, 220, 199, 243, 201, 241,
 200, 242, 177. 261, 178, 262,
 179, 263, 180, 264, 181, 265,
 198, 240, 202. 244, 219, 223,
 155, 281, 156. 282, 157, 283,
 158, 284, 159, 285, 160, 286,
 161, 287, 176. 260, 197. 239,
 182. 266, 203. 245, 218, 224 };

/* static int selected_features [] =
 { 245 }; */



   if (argc ≠ 6)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
      exit(1);
   }

   if ((file = fopen(argv[1], "r")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n". argv[1]);
   exit(0);
   }
   if ((outfile = fopen(argv[2], "w")) == NULL)
   {
   printf(stderr, "Couldn't open output file %s\n", argv[2]);
   exit(0);
   }
num_patterns = atoi(argv[3]);
num_features = atoi(argv[4]);
dimensions = atoi(argv[5]);

for (i=1; i ≤ num_patterns; i++)

  for (j=0; j ≤ num_features; j++)
    if (j==0) fscanf(file."%d", &junk);
else
   fscanf(file, "%f",&data[i][j]);
```

```
for(i=1; i≤ num_patterns; i++)
{
   fprintf(outfile,"%d ",class[i]);
   for(j=0; j< dimensions;j++)
      fprintf(outfile, "%f ",data[i][selected_features[j]]);
        fprintf(outfile,"\n");
}



fclose(file);
fclose(outfile);

}
/*********************************************

 Program:  addclassid.c


 Description:  Adds classid's to the
 feature set for use in LNKnet

*********************************************/

#include<stdio.h>

main(argc, argv)
int argc;
char *argv[];
{
   int i, j=0, done = 0, num_patterns, num_features;
   float val,data[800][800];
   char filename[80], temp[20];
   FILE *file, *outfile;

    static int class[] =

   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0. 0, 0, 0, 0, 0. 0, 0, 0, 0, 0, 0, 0, 0, 0. 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0. 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. 0, 0, 0,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1. 1, 1, 1, 1, 1, 1, 1. 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1. 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1. 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1. 1, 1, 1,
    1, 1, 1, 1. 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1. 1, 1, 1,
```

```
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3  };


  if (argc ≠ 5)
  {
    fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
    exit(1);
  }

  if ((file = fopen(argv[1], "r")) == NULL)
  {
  printf(stderr, "Couldn't open file list %s\n", argv[1]);
  exit(0);
  }
  if ((outfile = fopen(argv[2], "w")) == NULL)
  {
  printf(stderr, "Couldn't open output file %s\n", argv[2]);
  exit(0);
  }
num_patterns = atoi(argv[3]);
num_features = atoi(argv[4]);

for (i=0; i < num_patterns; i++)
 for (j=0; j < num_features; j++)
   fscanf(file, "%f",&data[i][j]);

for (i=0; i < num_patterns; i++)
 {
  if (i ≠ 0 ) fprintf (outfile, "\n");
  fprintf (outfile, "%d ", class[i]);
   for (j=0; j < num_features; j++)
     fprintf (outfile, "%f ", data[i][j]);

 }
fclose(file);
fclose(outfile);


}
```

```
/*************************************************

 Program:  removeclassid.c


 Description:  Removes classid's from the
 feature set.
*************************************************/



#include<stdio.h>

main(argc, argv)
int argc;
char *argv[];
{
   int  i, num_patterns, num_features, size;
   char filename[80], *buf, *ptr;
   FILE *file, *outfile;


   if (argc != 5)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
      exit(1);
   }

num_patterns = atoi(argv[3]);
num_features = atoi(argv[4]);

size = 15 * num_features + 2;

   if ((buf = (char *) malloc(size)) == NULL)
   {
      fprintf(stderr, "Couldn't allocate %d bytes of storage\n",size);
      exit(0);
   }
   if ((file = fopen(argv[1], "r")) == NULL)
   {
   fprintf(stderr, "Couldn't open file list %s\n", argv[1]);
   exit(0);
   }
   if ((outfile = fopen(argv[2], "w")) == NULL)
   {
   fprintf(stderr, "Couldn't open output file %s\n", argv[2]);
   exit(0);
   }

   ptr = buf+2;
   for (i=0; i < num_patterns; i++)
   {
```

```
        if (fgets(buf, size, file)==NULL)
        {
           fprintf(stderr, "PANIC!! Read error or EOF encountered\n");
           break;
        }
        fputs(ptr, outfile);
     }
     free(buf);
     fclose(file);
     fclose(outfile);

}


/*************************************************

Program: rand4c.c

Description:  Randomizes 4 classes of data, which
is in a file format ready for LNKnet, into
training and testing sets.  An equal number of
data samples are place in each set.


*************************************************/

#include<stdio.h>
#include<sys/types.h>
#include<sys/time.h>

main(argc, argv)
int argc;
char *argv[];
{

int j,x,i,k,idx[100][100],temp, num_patterns, num_features;
float data[500][500];
FILE *file, *outfile1, *outfile2, *outfile3;


   if (argc ≠ 7)
   {
      fprintf(stderr, "%s: usage: %s <infile> <outfile>\n", argv[0], argv[0]);
      exit(1);
   }

   if ((file = fopen(argv[1], "r")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n", argv[1]);
   exit(0);
   }
```

```
  if ((outfile1 = fopen(argv[2], "w")) == NULL)
  {
  printf(stderr, "Couldn't open output file %s\n", argv[2]);
  exit(0);
  }

  if ((outfile2 = fopen(argv[3], "w")) == NULL)
  {
  printf(stderr, "Couldn't open output file %s\n", argv[2]);
  exit(0);
  }
  if ((outfile3 = fopen(argv[6], "w")) == NULL)
  {
  printf(stderr, "Couldn't open output file %s\n", argv[6]);
  exit(0);
  }


/** read in the datafile **/

num_patterns = atoi(argv[4]);
num_features = atoi(argv[5]);

for (i=0; i < num_patterns; i++)
  for (j=0; j < num_features+1; j++)
  {
   if (j==0)
        fscanf(file, "%f", &data[i][j]);

   else
        fscanf(file, "%f",&data[i][j]):


 }


/** randomize a set of numbers for equal numbers of training and testing **/


for (i=0; i < 4; i++)
{
srandom((long) time(NULL));
for (j=0; j<100; j++)
   idx[i][j] = j;

for (j=0;j<100;j++)
  {
   x=random()%99;
   temp = idx[i][x];
   idx[i][x] = idx[i][j];
   idx[i][j] = temp;
  }
```

```
}
/** print values to the screen for grins **/
/* for (i=1; i<4; i++)
for (j=0; j<100; j++) */


/** write the data to train and test files **/
for (i=0; i<4; i++)     /* separate each class */
{
    if (i == 0)       /* class one */
      {
         for (j=0; j < 100; j++)    /* number of patterns in class one */
          {
            if (j≤49)       /* split 50-50 into test and train */
              {             /* writing to train file */

               if (j≠0) fprintf (outfile1, "\n");
                 for (k=0; k < num_features+1; k++)
                  {
                  if (k == 0) fprintf(outfile1, "%d ",
 (int) data[idx[i][j]][k]);

else
                  fprintf(outfile1, "%f ", data[idx[i][j]][k]);
                  }
               }
             if (j≥50)      /* writing to test file */
              {
                if (j≠50) fprintf(outfile2, "\n");
                  for (k=0; k < num_features+1; k++)
                   {
                  if (k == 0) {fprintf(outfile2, "%d ",
(int) data[idx[i][j]][k]);
 fprintf(outfile3,"%d\n",(idx[i][j]+1));}  /** print list to a file **/
else
                  fprintf(outfile2, "%f ", data[idx[i][j]][k]);

                    }
          }
            }
        }

 else
    if (i == 1)       /* same as above, but for the next class */
      {
         for (j=0; j < 100; j++)
          {
            if (j≤49)
              {
               fprintf (outfile1, "\n");
                 for (k=0; k < num_features+1; k++)
                  {
```

```
  /* must account for where the second class is located in the file */
                if (k == 0) fprintf(outfile1, "%d ",
(int) data[100+idx[i][j]][k]);
else
                fprintf(outfile1, "%f ", data[100+idx[i][j]][k]);
                 }
               }
              if (j≥50)
               {
                  fprintf(outfile2, "\n");
                   for (k=0; k < num_features+1; k++)
                    {
                   if (k == 0) { fprintf(outfile2, "%d ",
(int) data[100+idx[i][j]][k]);
fprintf(outfile3, "%d\n", (idx[i][j]+101));}

else
                fprintf(outfile2, "%f ", data[100+idx[i][j]][k]);

                  }
        }
          }
       }


 else
    if (i == 2)       /* same as above, but for the next class */
      {
         for (j=0; j < 100; j++)
          {
            if (j≤49)
             {
              fprintf (outfile1, "\n");
               for (k=0; k < num_features+1; k++)
                {
  /* must account for where the third class is located in the file */
                if (k == 0) fprintf(outfile1, "%d ",
(int) data[200+idx[i][j]][k]);
else
                fprintf(outfile1, "%f ", data[200+idx[i][j]][k]);
                 }
               }
              if (j≥50)
               {
                  fprintf(outfile2, "\n");
                   for (k=0; k < num_features+1: k++)
                    {
                   if (k == 0) { fprintf(outfile2, "%d ",
(int) data[200+idx[i][j]][k]);
fprintf(outfile3, "%d\n", (idx[i][j]+201)); }
else
                fprintf(outfile2, "%f ", data[200+idx[i][j]][k]);
```

```
                }
        }
          }
       }

else
   if (i == 3)       /* same as above, but for the next class */
     {
        for (j=0; j < 100; j++)
         {
           if (j≤49)
            {
             fprintf (outfile1, "\n");
              for (k=0; k < num_features+1; k++)
               {
 /* must account for where the fourth class is located in the file */
                if (k == 0) fprintf(outfile1, "%d ",
(int) data[300+idx[i][j]][k]);
else
                fprintf(outfile1, "%f ", data[300+idx[i][j]][k]);
               }
            }
           if (j≥50)
            {
              fprintf(outfile2, "\n");
               for (k=0; k < num_features+1; k++)
                {
               if (k == 0) { fprintf(outfile2, "%d ",
(int) data[300+idx[i][j]][k]);
fprintf(outfile3, "%d\n", (idx[i][j]+301));}

else
                fprintf(outfile2, "%f ", data[300+idx[i][j]][k]);

                }
         }
           }
       }



} /*end for loop*/
} /*end main*/

/*****************************************

  Program: sort_best_feature.c

  Description: This program is part
```

```
 of the script file, for add-on testing.
 It finds the best feature during addon
 testing.
                ****************************************/


#include <stdio.h>
main()
{

int i,count=0,feature_number[500];
float test_error[500];
FILE *infile, *outfile;


 if ((infile = fopen("test_features", "r")) == NULL)
    {
    printf(stderr, "Couldn't open file list %s\n");
    exit(0);
    }
    if ((outfile = fopen("feature", "a")) == NULL)
    {
    printf(stderr, "Couldn't open output file %s\n");
    exit(0);
}
for(i=0;i≤500;i++)
    test_error[i]=1000.00;


i=1;
while (fscanf(infile, "%d %f",&feature_number[i],&test_error[i]) ≠ EOF){
i++;
count++;
}
/*for(i=1;i≤ 8;i++){
fscanf(infile, "%d", &feature_number[i]);
fscanf(infile, "%f",&test_error[i]); */

/* count = 8; */


/*****************************************************************************
Sort
*****************************************************************************/
piksr2(count,test_error,feature_number);

fprintf(outfile,"%d\n", feature_number[1]);


}
```

```
/*****************************************

 Program: geterror.c

 Description: finds which patterns
 were misclassified.  This program is
 is part of the script file getmiss.

*****************************************/


#include <stdio.h>
#include <string.h>
main()
{
int j,how_many;
float train_err[100],test_err[100],train_rmserr[100],test_rmserr[100];
FILE *infile, *outfile;

   if ((infile = fopen("error", "r")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n", "miss" );
   exit(0);
   }
   if ((outfile = fopen("error_report", "w")) == NULL)
   {
   printf(stderr, "Couldn't open file list %s\n", "test.test" );
   exit(0);
   }
fscanf(infile,"%d", &how_many);

/* get the error */

  for (j=1; j<=how_many/4; j++)

     fscanf(infile, "%f %f %f %f" ,&train_err[j], &train_rmserr[j], &test_err[j], &test_rmserr[j]);

piksr2(how_many/4,test_err,train_err);
piksr2(how_many/4, test_rmserr,train_rmserr);

fprintf(outfile, "%2.2f %2.2f %2.2f %2.2f", test_err[1],train_err[1], test_rmserr[1], train_rmserr[1]);

} /*end main*/
```

Bibliography

1. Bertille, J.M. and A.E. Yacoubi. "Global Cursive Postal Code Recognition Using Hidden Markov Models." *Proceedings from the First European Conference dedicated to Postal Technologies1*. 129–137. 1993.
2. Bush, Larry F. *The Design of an Optimum Alphanumeric Symbol Set for Cockpit Displays*. MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1977.
3. Cohen, E. et al. "Understanding Spatially Structured Handwritten Text." *First International Conference on Document Analysis and Recognition2*. 984–992. 1991.
4. Davenport, Wilbur B. *Probability and Random Processes*. New York, NY: McGraw-Hill, 1987.
5. Duda and Hart. *Pattern Recognition and Scene Classification*. New York, NY: McGraw-Hill, Inc., 1971.
6. Fukunaga, Keinosuke. *Introduction to Statistical Pattern Recognition*. San Diego, CA: Academic Press, Inc., 1990.
7. Garcia, G.L. "The State of the Art of Handwriting Recognition for Automated Mail Processing." *Proceedings from the First European Conference dedicated to Postal Technologies1*. 437–446. 1993.
8. Gaskill, Jack D. *Linear Systems, Fourier Transforms, and Optics*. New York, NY: John Wiley and Sons, 1978.
9. Gilloux, Michel and Manuel Leroux. "Recognition of Cursive Script Amounts on Postal Cheques." *Proceedings from the First European Conference dedicated to Postal Technologies2*. 705–712. 1993.
10. Ho, T.K. et al. "Word Recognition With Multi-Level Contextual Knowledge." *First International Conference on Document Analysis and Recognition2*. 905–915. 1991.
11. Kabrisky, Matthew. *A Proposed Model for Visual Information Processing in the Human Brain*. Chicago, IL: University of Illinois Press, 1964.
12. Kukolich, Linda, "LNKnet," 1992.
13. Moreau, J.V. et al. "A Postal Check Reading System." *First International Conference on Document Analysis and Recognition2*. 758–764. 1991.
14. New Mexico, University of, "Visual Programming System and Software Development Environment for Data Processing and Visualization (Khoros)," 1991.
15. of Excellence for Document Analysis, Center and Recognition, "USPS Office of Advanced Technology Database of Handwritten Cities, States, ZIP Codes, Digits, and Alphabetic Characters," 1992.
16. O'Hair, Mark A. *A Whole Word and Number Reading Machine Based on Low Frequency Fourier Complex and Amplitude Spectrums*. MS thesis, AFIT/GEO/ENG/84D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.

17. O'Hair, Mark A. *A Whole Word and Number reading Machine Based on 2-Dimensional Low Frequency Fourier Transforms*. PhD dissertation, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

18. Parsons, Thomas W. *Voice and Speech Processing*. New York, NY: McGraw-Hill, Inc., 1987.

19. Pearson, E.S. and H.O. Hartley. *Biometrika Tables for Statisticians*. Cambridge, England: Cambridge University Press, 1966.

20. Pintsov, L.A. "Handwritten Character Recognition Some Observations Concerning Principles and Modus Operandi." *Proceedings from the First European Conference dedicated to Postal Technologies1*. 26–34. 1993.

21. Powalka, R.K., et al. "A Toolbox for Recognition of Varied Handwritten Script." *Proceedings from the First European Conference dedicated to Postal Technologies1*. 140–147. 1993.

22. Radoy, Charles. *Pattern Recognition by Fourier Series Transformations*. MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, August 1968.

23. Rogers, Steven K. and others. *An Introduction to Biological and Artificial Neural Networks*. Bellingham, WA: SPIE Optical Engineering Press, 1991.

24. S., Tsujimoto and Asanda H. "Resolving Ambiguity in Segmenting Touching Characters." *First International Conference on Document Analysis and Recognition2*. 701–709. 1991.

25. S., Zeki. "The Visual Image in Mind and Brain," *Scientific American*, *267*:69–76 (Sept 1992).

26. Simon, J.C. "On The Robustness of Recognition of Degraded Line Images." *Proceedings from the First European Conference dedicated to Postal Technologies2*. 695–696. 1993.

27. Srihari, S.N., et al. "Interpretation of Handwritten Addresses in US Mail Stream." *Proceedings from the First European Conference dedicated to Postal Technologies1*. 421–428. 1993.

28. Suarez, P.F. *Face Recognition with the Karhunen-Loeve Transform*. MS thesis, AFIT/GE/ENG/91D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

29. Tallman, Oliver H. *The Classification of Visual Images by Spatial Filtering*. PhD dissertation, Air Force Institute of Technology, 1969.

30. Tou, Julius T. and Rafael C. Gonzalez. *Pattern Recognition Principles*. Reading, MA: Addison-Wesley Publishing, 1974.

Vita

Captain Gary F. Shartle was born on 28 March 1965 in Fullerton, California and graduated from Cumberland Valley High School in Mechanicsburg, PA in June, 1983. Upon graduating from Grove City College in 1987 with a Bachelor of Science in Electrical Engineering, he entered the Air Force, and was an operations management officer assigned to the 69th Bombardment Squadron at Loring AFB in Maine. In 1991, to complete the operational tour, he was assigned to the 3rd Tactical Fighter Wing at Clark AFB in the Phillippines, serving as a command and control officer. Due to the fiery explosion of Mt. Pinatubo, he was transferred to Wright-Patterson AFB. In May 1992, Captain Shartle entered the Air Force Institute of Technology at Wright-Patterson AFB, Ohio to pursue a Master of Science degree in Electro-Optics.

Permanent address: 4184 Tonawanda Trail
Beavercreek, OH 45430

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE
HANDWRITTEN WORD RECOGNITION
BASED ON
FOURIER COEFFICIENTS

5. FUNDING NUMBERS

6. AUTHOR(S)
Gary F. Shartle

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GEO/ENG/93D-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Major Mark O'Hair
WL/AAR
WPAFB, OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A machine which can read unconstrained words remains an unsolved problem. For example, automatic entry of handwritten documents into a computer is yet to be accomplished. Most systems attempt to segment letters of a word and read words one character at a time. Segmenting a handwritten word is very difficult and often, the confidence of the results is low. Another method which avoids segmentation altogether is to treat each word as a whole. This research investigates the use of Fourier Transform coefficients, computed from the whole word, for the recognition of handwritten words. To test this concept, the particular pattern recognition problem studied consisted of classifying four handwritten words, 'Buffalo', 'Vegas', 'Washington', 'City.' Several feature subsets of the Fourier coefficients are examined. The best recognition performance of 76.2% was achieved when the Karhunen-Loeve transform was computed on the Fourier coefficients.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Pattern Recognition, Recognition, Whole Word Recognition	110
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL